НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій, СТІРЕНКО

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система електронного забезпечення науково-методичної діяльності вищого навчального закладу»

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-64
Говрас Вячеслав Сергійович _____

Керівник:
Доцент, доктор технічних наук
Клименко Ірина Анатоліївна _____

Консультант з нормконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
Доцент, кандидат технічних наук
Писаренко Андрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*

Кафедра *Обчислювальної техніки*

Рівень вищої освіти – перший (бакалаврський)

Спеціальність *121 Інженерія програмного забезпечення*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *О, А, Павлов*
(підпис) (ініціали, прізвище)
“___”_____2020 р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Говраса Вячеслава Сергійовича

1. Тема проекту «*Система електронного забезпечення науково-методичної діяльності вищого навчального закладу*»
керівник проекту *професор кафедри ОТ, д.т.н., доц.Клименко І.А.*

затверджена наказом по університету від “21” квітня 2020 р. №1393
2. Термін подання студентом проекту «07» *червня 2020 року*
3. Вихідні дані до проекту *технічна документація, статистичні та теоретичні дані*

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Методика тестування

4) Керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо

_____

_____

_____

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1.	д.т.н., доц.Клименко І.А.		
2.	д.т.н., доц.Клименко І.А.		
3.	д.т.н., доц.Клименко І.А.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

N з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Затвердження теми роботи	15.12.2019	
2	Вивчення та аналіз завдання	20.12.2019-02.02.2020	
3	Проектування програмного	02.02.2020-01.03.2020	
	забезпечення		
4	Програмна реалізація продукту	01.03.2020-15.04.2020	
5	Оформлення пояснювальної записки	*15.04.2020-13.05.2020*	
6	Захист програмного продукту	*13.05.2020-19.05.2020*	
7	Перед захист	*18.04.2020*	
8	Захист	*16.05.2020*	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В дипломній роботі бакалавра запропонована та реалізована «Система електронного забезпечення науково-методичної діяльності вищого навчального закладу» Розроблена система дозволяє зберігати та опрацьовувати науково-методичну діяльність. Схема розробленої системи ілюструється. Наведені теоретичні та експериментальні оцінки ефективності реалізованої системи. Веб додаток дозволяє обробляти велику кількість наукових статей, методичних вказівок, підручників. Додаток було створено на платформі node.js *(React)* та *PHP* з використанням реляційної бази даних *Postgres.*

АННОТАЦИЯ

В дипломной работе бакалавр предложил и реализовал «систему электронного поддержания научно-методической деятельности высшего учебного заведения», разработана система, позволяющая сохранять и обрабатывать научно-методическую деятельность. Схема разработанной системы иллюстрируется. Представлены теоретические и экспериментальные оценки эффективности реализованной системы. Веб-приложение позволяет обрабатывать большое количество научных статей, методических рекомендаций, учебников. Приложение было создано на уделе *JS (React)* и платформе *PHP* с использованием реляционной базы данных *Postgres.*

ANNOTATION

In the diploma thesis the bachelor proposed and implemented "the system of electronic maintenance of scientific and methodical activity of higher education institution", developed a system that allows to save and process scientific and methodical activity. The scheme of the developed system is illustrated. Theoretical and experimental estimates of the implemented system's efficiency are presented. The web-application allows processing many scientific articles, methodical recommendations, and textbooks. The application was created on JS (React) and PHP platform using a relational Postgres database

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А 4		Завдання на	2	
			дипломний проект		
2	А4	ІАЛЦ.467800.001 ТОП	Опис проекту	1	
3	А4	ІАЛЦ.467800.001 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.467800.002 ПЗ	Пояснювальна	44	
			Записка		
5	А3	ІАЛЦ.467800.003 Д1	Схема моделі даних	1	
6	А4	ІАЛЦ.467800.004 Д2	Схема алгоритму	1	
			виконання скрипту		
7	А3	ІАЛЦ.467800.005 Д3	Схема виконання	1	
			послідовності		
			Скрипту		

ЗМІСТ

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Назва розробки: Система електронного забезпечення науково-методичної діяльності вищого навчального закладу.

Область застосування: науково-методична діяльність.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки даного проекту стало завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ

Метою даного проекту є розробка електронної системи, яка дозволить легко керувати науково-методичною діяльністю.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, довідники, технічна документація публікації в виданнях та Інтернеті, що містять опис архітектури і принципи розробки такого програмного забезпечення.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Матеріал має відповідати наступним критеріям:

- Достатній об’єм інформації;
- Матеріал повинен бути поданий у максимально зрозумілій формі.

5.2 Вимоги до програмного забезпечення:

- Операційна система *Linux/Windows/macOS/Android/iOS;*
- Підключення до Інтернету
- Браузер.

6. ЕТАПИ РОЗРОБКИ

Вивчення джерел за тематикою роботи	Дата
Розроблення і узгодження технічного завдання	
Моделювання структури продукту	
Розробка продукту	
Тестування продукту	
Виправлення помилок	
Оформлення документації дипломної роботи	

Пояснювальна записка
до дипломного проекту

на тему: Система електронного забезпечення науково-методичної діяльності вищого навчального закладу

Київ – 2020

ЗМІСТ

РОЗДІЛ 1
ОГЛЯД СИСТЕМИ ЕЛЕКТРОННОГО ЗАБЕЗПЕЧЕННЯ НАУКОВО-МЕТОДИЧНОЇ ДІЯЛЬНОСТІ ВИЩОГО НАВЧАЛЬНОГО ЗАКЛАДУ І АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ СИСТЕМ НАВЧАННЯ

1.1Огляд існуючих систем

Для вирішення цієї проблеми було створено систему, яка схожа на системи документообігу та адаптована під наші вимоги. Наша система електронного забезпечення науково-методичної діяльності має забезпечувати можливість легко зберігати та швидко робити пошук по всій бібліотеці.

Спочатку було проведено аналіз існуючих рішень на ринку але реалізацій, які б нам підходили, не було знайдено. Тому було вирішено розробити свою реалізацію.

Серверна частина

Для реалізації серверної частини було вирішено використати мову PHP. В першу чергу PHP було обрано саме тому, що ця технологія вже давно на ринкові. Люди знають як з нею працювати, тому легко знайти розробника, якщо це необхідно.

Для чого використовується PHP[1]? В основному все, що ми хочемо зробити на веб-сервері, ми можемо виконати з PHP. Зробити блог? Так. Створити повноцінну програму as-a-serivce? Абсолютно. Написати невеликий сценарій, щоб обробити деякі дані за кілька секунд? PHP чудово підходить для цього. Написати складний набір сценаріїв.

В нашому випадку ми використовуємо PHP для реалізації API, яке потім використовуємо на SPA[2] (single page application).

Також це доволі швидке рішення. PHP розвивається цікавими шляхами. Вона зростає до того, щоб бути досить повнофункціональною об'єктно-орієнтованою мовою, схожою на Java (для кращого чи гіршого). І подібно до Java, вона отримує легкі абстракції для функціонального програмування. Зростає також неабиякий набір інструментів - PHP використовує Composer, що дає можливість щоб усі ці великі проекти з відкритим кодом у PHP працювали разом трохи краще.

Також ми не повинні забувати про підвищення швидкості, яку PHP зробив у серії релізів PHP 7, це вважається ініційованим HHVM, який вийшов з Facebook. Спочатку існував ризик, що швидкість HHVM[1] зірве спільноту PHP. Але це не відбулось. Натомість PHP просто стає набагато швидше.

1.2 Вибір бази даних

Для вирішення проблеми із зберіганням даних було обрано *SQL* базу даних замість *No SQL.* Так, як наші дані легше зберігати та обробляти в реляційному вигляді. Детальніше описано нижче.

Масштабованість

Подумайте про високу будівлю у вашому районі. Якщо є можливість, що було б краще - додати більше поверхів у цій будівлі чи створити нову будівлю для більшої кількості мешканців?

Це проблема для баз даних *SQL* і *No SQL*. Бази даних *SQL* є вертикально масштабованими. Це означає, що навантаження на одному сервері можна збільшити за рахунок збільшення таких речей, як оперативна пам’ять, процесор або *SSD.* (До цієї будівлі можна додати більше поверхів). З іншого боку, бази даних *No SQL* є горизонтально масштабованими. Це означає, що більшу кількість трафіку можна обробляти за рахунок посилення або

додавання більшої кількості серверів у вашу базу даних *No SQL*. (До мікрорайону можна додати більше будівель).

Зрештою, краще додати більше будівель, ніж поверхів, оскільки це стійкіше (менше шансів створити Пізанську вежу !!!). Таким чином, *No SQL* може в кінцевому рахунку стати більшим і потужнішим, що робить бази даних *No SQL* кращим вибором для великих або постійно змінних наборів даних.

Дизайн схеми

Схема бази даних *SQL* і бази даних *No SQL* помітно відрізняється. Ця різниця в схемі робить реляційні бази даних *SQL* кращим варіантом для додатків, які потребують транзакцій, таких як система обліку, або для застарілих систем, побудованих для реляційної структури. А бази даних *No SQL* набагато краще підходять для великих даних, оскільки гнучкість є важливою вимогою, яку виконує їх динамічна схема.

Підтримка спільноти

SQL - це зріла технологія і багато досвідчених розробників це розуміють. Також велика підтримка доступна для всіх баз даних *SQL* від їхніх постачальників. Існує навіть багато незалежних консультантів, які можуть допомогти з базою даних *SQL* для дуже масштабного розгортання.

З іншого боку, *No SQL* порівняно нові. Тому деякі бази даних *No SQL* залежать від підтримки спільноти. Крім того, для налаштування та розгортання широкомасштабних систем *No SQL* доступні лише обмежені зовнішні експерти.

1.3 Контейнеризація системи (*Docker*)

Що таке Докер?

Docker - платформа контейнерної роботи з відкритим кодом[5]. Docker дозволяє розробникам пакувати програми в контейнери - стандартизовані виконувані компоненти, що поєднують вихідний код програми з усіма бібліотеками операційної системи (ОС) та залежностями, необхідними для запуску коду в будь-якому середовищі.

Хоча розробники можуть створювати контейнери без Docker, але з Docker, простіше, безпечніше створювати та розгортати, здійснювати керування контейнерами. По суті, це набір інструментів, який дозволяє розробникам створювати, розгортати, запускати, оновлювати та зупиняти контейнери за допомогою простих команд та автоматизації роботи.

Docker також відноситься до Docker, Inc[6]., компанії, що продає комерційну версію Docker, і до проекту з відкритим кодом Docker, в який вкладають Docker Inc. та багато інших організацій та осіб.

Навіщо використовувати контейнери?

Контейнери стають можливими завдяки ізоляції та віртуалізації процесів операційної системи (ОС), які дозволяють декільком компонентам додатків спільно використовувати ресурси одного примірника ядра ОС точно так же, як машинна віртуалізація дозволяє декільком віртуальним машинам (ВМ) спільно використовувати ресурси одного апаратного сервера.

Контейнери пропонують всі переваги віртуальних машин, включаючи ізоляцію додатків, економічну масштабованість і можливість утилізації. Однак додатковий рівень абстракції (на рівні ОС) дає важливі додаткові переваги:

- **Легка вага:** На відміну від *VMs*, контейнери не несуть корисного навантаження на весь екземпляр *ОС* - в них включені тільки процеси і залежно *ОС*, необхідні для виконання коду.
- **Велика ефективність використання ресурсів:** [4]За допомогою контейнерів можна запускати в кілька разів більше копій програми на одному і тому ж обладнанні, ніж за допомогою *VMs*. Це може знизити ваші витрати на хмару.
- **Підвищення продуктивності розробників:** У порівнянні з *VMs* контейнери швидше і простіше в розгортанні, ініціалізації та перезапуску. Це робить їх ідеальними для використання в трубопроводах безперервної інтеграції та безперервної доставки (CI / CD) і краще підходять для команд розробників, які використовують *Agile* і *DevOps*.

Як використовувати Docker images

Образи докерів містять виконуваний вихідний код програми, а також всі інструменти, бібліотеки і залежності, необхідні для запуску коду програми в якості контейнера. Коли ви запускаєте образ Docker, він стає одним екземпляром (або декількома екземплярами) контейнера.

Можна побудувати образ Docker з нуля, але більшість розробників витягують його з звичайних репозиторіїв. Кілька образів докера можна створити з одного базового образу, і вони будуть мати спільні риси стека.

Docker images складаються з шарів (*layers*), і кожен шар відповідає версії зображення. Всякий раз, коли розробник вносить зміни в зображення, створюється новий верхній шар, і цей верхній шар замінює попередній верхній шар як поточну версію зображення. Попередні шари зберігаються для відкатів або для повторного використання в інших проектах.

Кожен раз при створенні контейнера з зображення *Docker* створюється ще один новий шар під назвою контейнерний шар. Зміни, внесені в контейнер - такі як додавання або видалення файлів - зберігаються тільки на шарі-контейнері і існують тільки під час роботи контейнера. Цей ітеративний процес створення образів дозволяє підвищити загальну ефективність, оскільки кілька примірників контейнерів в реальному часі можуть працювати тільки з одного базового образу, і при цьому вони використовують загальний стік.

ВИСНОВОК ДО РОЗДІЛУ 1

В даному розділі було оглянуто систему KPI CONECT та пошук альтернатив на ринку. Так як альтернатив з даним функціоналом не має було прийнято рішення створити свою систему, яка буде відповідати таким вимогам:

- При потребі масштабуватись
- Можливість легко вносити зміни в реалізацію, додавати нові функціональні можливості при потребі
- Швидкий інтерфейс взаємодії

Також було розглянуто стек використаних технологій та порівняно з існуючим на ринку, В даному розділі було детально розглянуто технології на яких побудована система:

- Php (мова серверної розробки);
- Symfony (фреймворк для прискорення розробки на php);
- Docker (система контейнеризації яка дозволить легко розгорнути систему на віддаленому сервері);
- Postgres (реляційна база даних);

Наведені приклади існуючих технологій, та показано, чому був обраний цей стек технологій.

В даному проекті було поставлено завдання розробити систему електронного забезпечення науково-методичної діяльності вищого навчального закладу, яка буде реалізувати наступні функції:

- Створення нової персони якій будуть надаватись права на взаємодію з системою;
- Можливість заповнення інформації, яка з нею пов’язана (ім’я, контактні дані, інформація про наукову діяльність);
- Надати можливість додати до створеної персони нові наукові видання які можна буде переглядати, редагувати;

- Додати можливість створення навчально-методичних видань і їхня обробка;

Розробити легкий, зрозумілий інтерфейс та задокументувати його функціональні можливості.

РОЗДІЛ 2
ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ WEB ДОДАТКІВ ТА ПОРІВНЯННЯ З ІСНУЮЧИМИ

2.1 Протокол *HTTP*

HTTP-це протокол, який дозволяє отримувати ресурси, такі як документи HTML. Це основа будь-якого обміну даними в Інтернеті. HTML ‘є клієнт-серверним протоколом, тобто запити ініціює одержувач, як правило, веб-браузером. [7]Повний документ виходить з отриманих документів, таких як текст, описи макетів за допомогою розмітки, зображення, відео, скрипти (js) тощо.

На рис. 2.1 показано як пристрій взаємодіє з сервером по протоколу http через інтернет

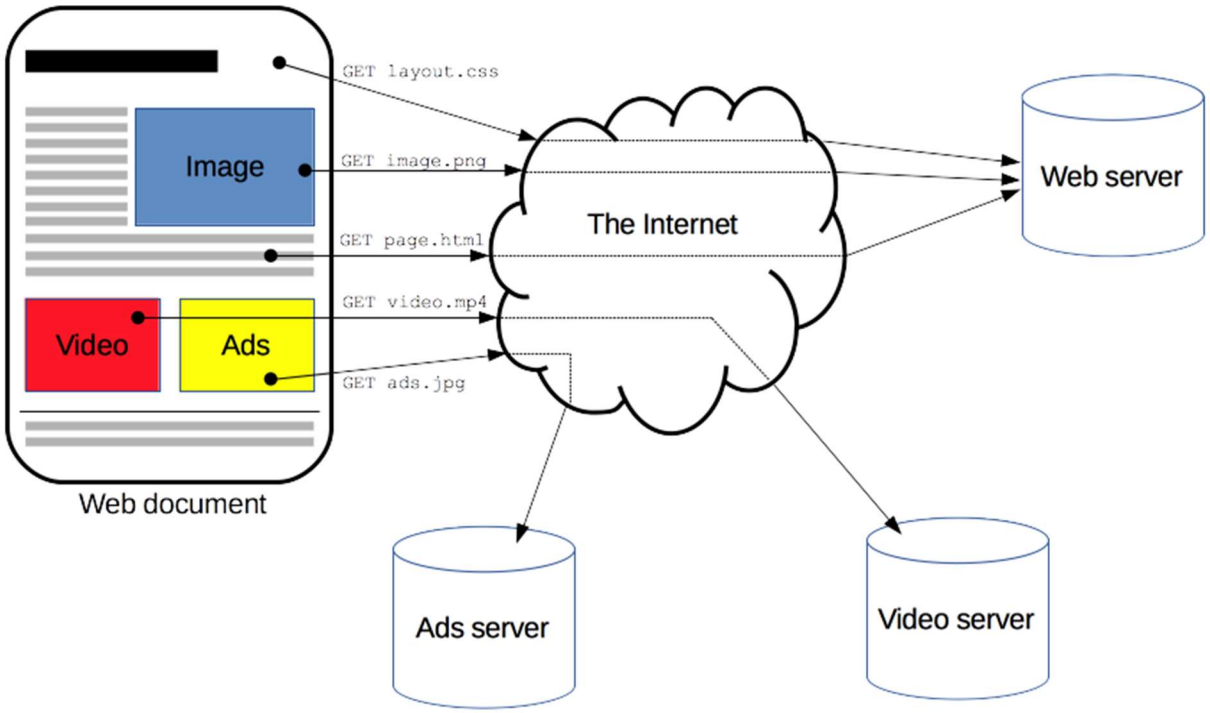


Рис 2.1 Схема роботи *HTTP* протоколу

Клієнти і сервери спілкуються за допомогою обміну окремими повідомленнями (на відміну від потоку даних). Повідомлення, відправлені клієнтом, зазвичай веб-браузером, називаються запитами, а повідомлення, що відправляються сервером як відповідь, називаються відповідями.

HTTP Розроблений на початку[7] 1990-х років, HTTP є розширюваним протоколом, який розвивався з плином часу.

На рис. 2.2 показано структуру веб інтерфейсу де показано транспортні рівні, та де знаходиться *HTTP*

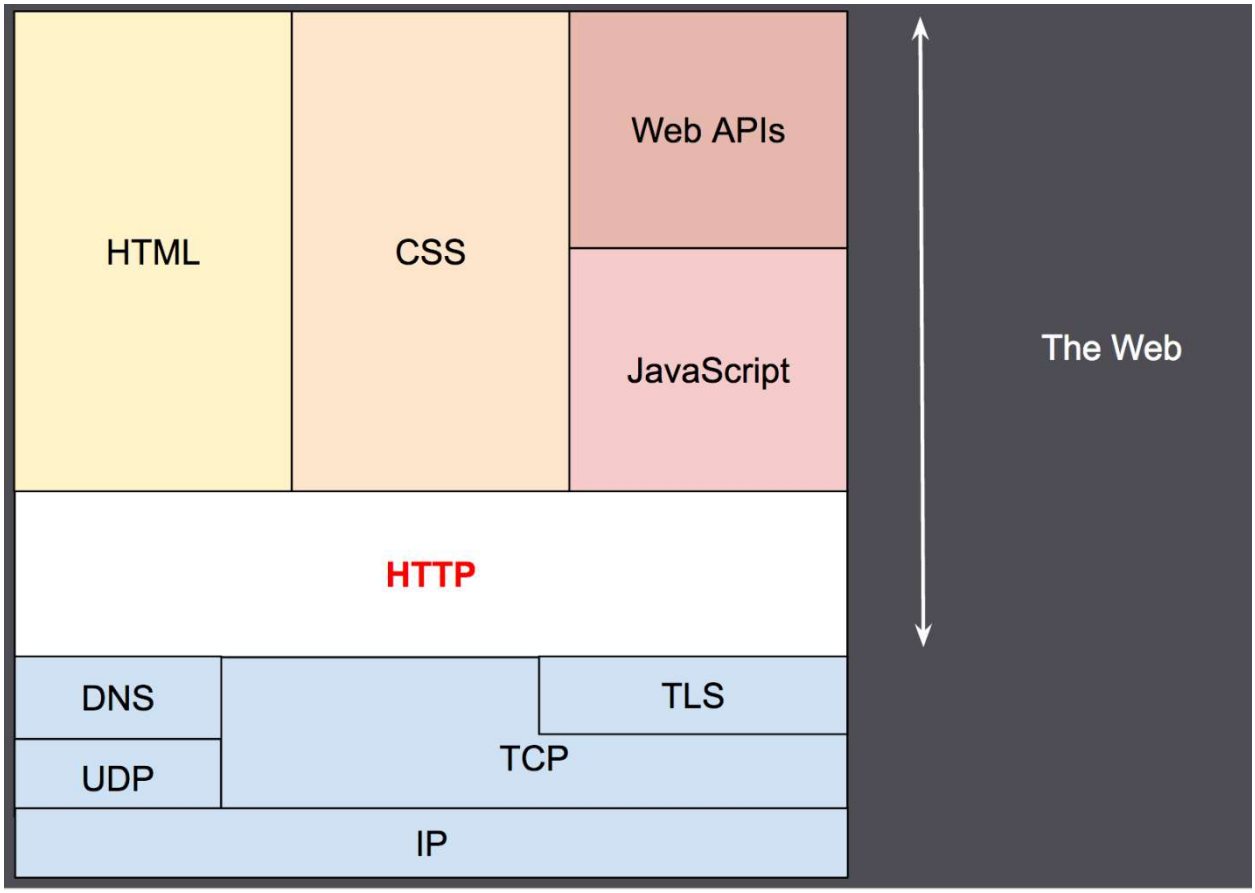


Рис. 2.2 Структура *Web*

Це протокол прикладного рівня, який передається по TCP або по TLS-зашифрованого TCP-з'єднання, хоча теоретично може використовуватися будь-який надійний транспортний протокол.

Завдяки своїй розширюваності, він використовується не тільки для отримання гіпертекстових документів, а й зображень і відео, або для відправки контенту на сервери, як і в випадку з *HTML*-формою результатів. *HTTP* також може використовуватися для отримання частин документів для поновлення веб-сторінок на вимогу.

Компоненти систем на базі HTTP

HTTP э клієнт-серверним протоколом: запити посилає одна сутність, користувач-агент (або проксі від його імені). У більшості випадків користувач-агент є веб-браузером, але це може бути що завгодно, наприклад, робот, який повзає по Мережі, щоб заповнити і підтримувати індекс пошукової системи.

Кожен окремий запит надсилається на сервер, який обробляє його і надає відповідь. Між клієнтом і сервером існує безліч сутностей, так званих *proxy*, які виконують різні операції і виступають, наприклад, в якості шлюзів або кешем.

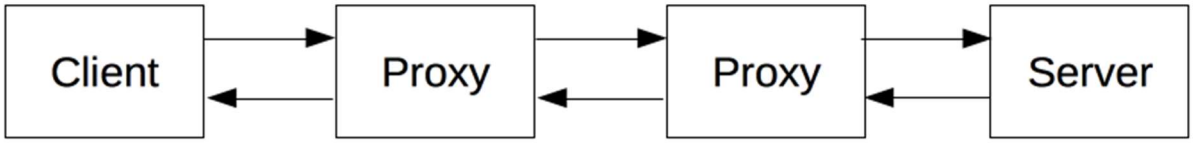


Рис. 2.3 Ілюстрація роботи *proxy* в *HTTP*

В реальності між браузером і сервером, що обробляють запит, знаходиться більше комп'ютерів та маршрутизаторів, модемів та багато іншого. Завдяки багаторівневому дизайну *Web*, вони приховані в мережевому і транспортному рівнях. *HTTP* знаходиться зверху, на прикладному рівні. Хоча це і важливо для діагностики мережевих проблем, основні рівні в основному не мають відношення до опису *HTTP*.

HTTP це просто

HTTP, як правило, спроектований так, щоб бути простим і легким для читання людиною, навіть з доданою складністю, введеної в *HTTP* / 2 шляхом інкапсуляції *HTTP*-повідомлень у фрейми. Повідомлення *HTTP* можуть бути прочитані і зрозумілі людиною, що спрощує тестування для розробників, а також знижує складність для новачків.

HTTP є легко розширюваним

Введені в *HTTP* / 1.0, заголовки *HTTP* дозволяють легко розширювати цей протокол і експериментувати з ним. Новий функціонал може бути навіть просто угодою між клієнтом і сервером про семантику нового заголовка.

HTTP не має стану, але з сеансом

HTTP є апатридом: немає зв'язку між двома запитами, які виконувались послідовно по тому самому з'єднанню.

Це відразу ж може спричинити проблеми для користувачів, які намагаються взаємодіяти з певними сторінками узгоджено, наприклад, використовуючи кошика для покупок в електронній торгівлі. Але в той час як ядро HTTP само по собі є апатридом, HTTP-Cookies дозволяють використовувати сеанси зі зворотним зв'язком. Використовуючи розширюваність заголовків, HTTP-Cookies додаються в робочий процес, дозволяючи створювати сеанси по кожному HTTP запиту в одному і тому ж контексті, або в одному і тому ж стані.

2.2 Огляд платформи *Node.js*

Особливості середовища Note.js

Node.js - це крос-платформне середовище виконання *JavaScript* з відкритим кодом, яке виконує *JavaScript* [8] за межами веб-браузера.

На основі нього працює більшість існуючих таск-ранерів про, які буде йти мова нижче. *Node.js* []дозволяє розробникам використовувати *JavaScript* для написання різних консольних інструментів та для сценаріїв на стороні сервера, запуску скриптів на стороні сервера для створення динамічного вмісту веб-сторінки перед відправкою сторінки у веб-браузер користувача. Отже, *Node.js* представляє парадигму *"JavaScript* скрізь", що об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для скриптів на сервері та клієнтах.

Хоча *.js* є [8] стандартним розширенням імені файлу коду *JavaScript,* ім'я *"Node.js"* не посилається на конкретний файл у цьому контексті і є лише назвою платформи. *Node.js* має керовану архітектуру подій яка, здатна до асинхронного вводу / виводу.

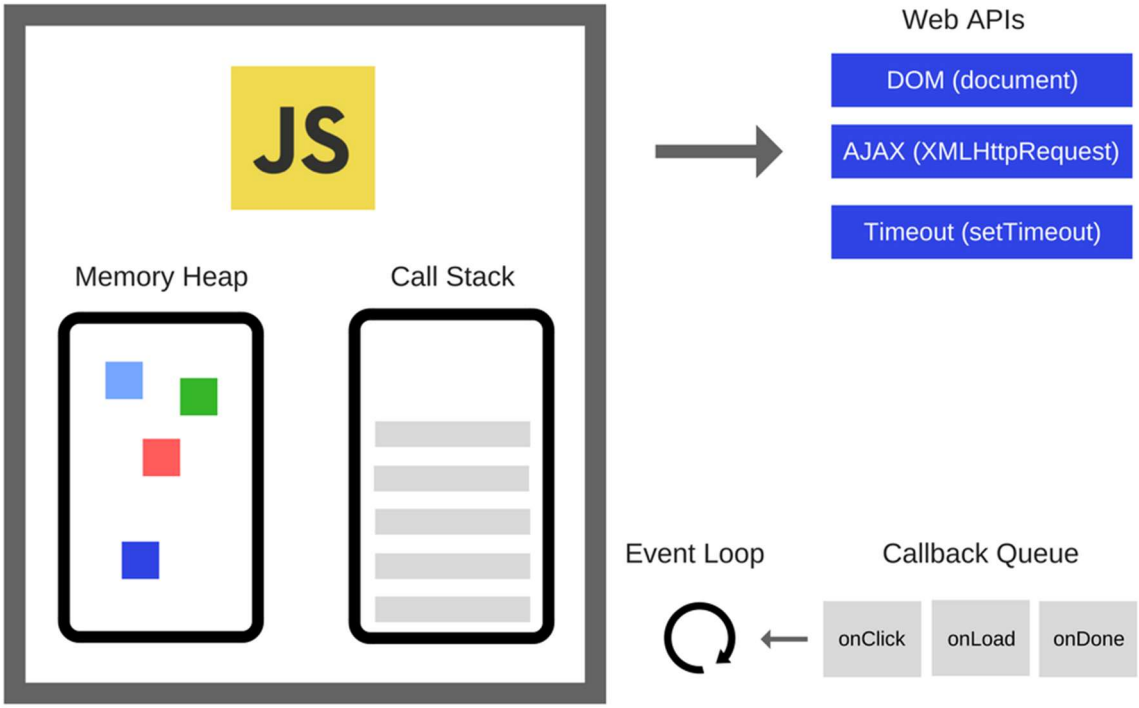


Рис. 2.4 Ілюстрація роботи асинхронного вводу\виводу в JS\Node.js

Ці варіанти дизайну мають на меті оптимізувати пропускну здатність та масштабованість у веб-додатках із багатьма операціями вводу / виводу, а також для веб-додатків у режимі реального часу (наприклад, програм зв’язку в реальному часі та браузерних ігор).

Розроблений проект *Node.js* раніше керувався Фондом *Node.js*, і тепер об'єднався з Фондом *JS*, щоб утворити Фонд *Open JS*, що сприяє програмі спільних проектів *Linux Foundation*.

Історія створення платформи node.js

Node.js був написаний спочатку Райаном Далом у 2009 році, приблизно через тринадцять років після введення першого середовища на сервері *JavaScript, LiveWire Pro Web Netscape*. Початковий реліз підтримував лише

Linux та *Mac OS X*. Його розробкою та обслуговуванням керував Дал, а пізніше спонсорував *Joyent.*

Дал розкритикував обмежені можливості найпопулярнішого веб-сервера 2009 року, *Apache HTTP Server*, який не міг тримати велику кількість одночасних з'єднань та найпоширеніший спосіб створення коду (послідовне програмування), коли код або блокував весь процес, або використовував багато потокову архітектуру, яка спричиняла потокове глодання при великій кількості запитів.

8 листопада 2009 Дал продемонстрував проект року на європейській *Js conf. Node.js* поєднав в собі механізм *JavaScript* та *V8* від *Google*, цикл подій та *API* вводу / виводу низького рівня. У січні 2010 року було запроваджено менеджер пакетів для середовища *Node.js* під назвою *npm.* Менеджер пакетів полегшує розробникам публікацію та обмін готовими рішеннями для *Node.js* і призначений для спрощення встановлення, оновлення та видалення пакетів.

У червні 2011 року *Microsoft* та *Joyent* впровадили вбудовану версію *Windows Node.js.* Перша збірка *Node.js,* що підтримує *Windows,* була випущена в липні 2011 року.

У травні 2018 року один з оригінальних авторів *Node.js,* Райан Дал, випустив першу версію *Deno,* нову альтернативу Node.js на основі *TypeScript. Deno*, написаний на *Rust*, був розроблений для усунення недоліків *Node.js.*

Можливості node.js

Node.js дозволяє створювати веб-сервери та мережеві інструменти за допомогою *JavaScript* та колекцію "модулів", які керують різними функціональними можливостями. Модулі передбачені для вводу / виводу файлової системи, мереж (*DNS, HTTP, TCP, TLS / SSL* або *UDP*), бінарних даних (буферів), функцій криптографії, потоків даних та інших основних функцій. Модулі *Node.js* використовують *API*, призначений для зменшення складності серверних програм.

JavaScript є єдиною мовою, яку *Node.js* підтримує на власній основі, але доступно багато мов які компілюються в *JS.* Як результат, можна писати на *CoffeeScript, Dart, TypeScript, ClojureScript* та інші.

Node.js в основному використовується для побудови мережевих програм, таких як веб-сервери. Найбільш істотна відмінність *Node.js* від *PHP* полягає в тому, що більшість функцій у *PHP* блокуючі, тоді як функції *Node.js* не блокуючі (команди виконуються одночасно або навіть паралельно, і використовувати зворотні виклики для сигналізації завершення чи відмови).

Node.js офіційно підтримується в *Linux, macOS* та *Microsoft Windows* і *Server 2012* (і пізніших версіях). З підтримкою рівня 2 для *SmartOS* та *IBM AIX* та експериментальною підтримкою для *FreeBSD. OpenBSD* також працює, і версії *LTS* доступні для *IBM* і (*AS / 400*). Наданий вихідний код також може бути побудований на аналогічних операційних системах, які офіційно підтримуються або змінюються третіми сторонами для підтримки інших, таких як *NonStop OS* та серверів *Unix.*

2.3 Менеджери виконання сценаріїв (*Task Runners*)

Раніше *web* не розвивався так стрімко і в інтернеті ми користувалися сайтами лише для пошуку текстової інформації. Однак час змінився, і тепер розповсюдження вашого коду *JavaScript* може бути складним завданням. Ця проблема посилилась із збільшенням одно-сторінкових додатків (*SPA*). Вони, як правило, пишуться з використанням великих бібліотек.

Інструмент Make для мови C

Make йде з минулого, він був створений в 1977 році. Хоча це старий інструмент, він залишається актуальним. *Make* дозволяє писати окремі завдання для різних цілей. Наприклад, у вас можуть бути різні завдання для створення, мінімізації *JavaScript* та запуску тестів. Цю ідею ви можете знайти в багатьох інших інструментах.

Навіть незважаючи на те, що *Make* в основному використовується в проектах *C,* він жодним чином не пов'язаний з *C*. Джеймс Коглан докладно обговорює, як використовувати *Make* з *JavaScript.*

Завдання в npm

Ці сценарії можуть бути перелічені за допомогою *npm run*, а потім виконані за допомогою *npm run script*. Також можна називати простір імен своїми сценаріями, використовуючи такий конвент, як тест: *watch*. Проблема такого підходу полягає в тому, що він дбає про те, щоб він був крос-платформним. Замість *rm -rf* ви, ймовірно, хочете використовувати такі утиліти, як *rimraf* тощо. Тут можна викликати інші таск-ранери, щоб приховати той факт, що вони використовуються. Таким чином можна переробляти інструменти, зберігаючи інтерфейс таким самим.

Grunt

Grunt був першим відомим виконавцем завдань для розробників *frontend*. Його архітектура плагінів сприяла популярності. Плагіни часто складні самі по собі. В результаті, коли конфігурація зростає, може бути важко зрозуміти, що відбувається. Коли завдання запускається, воно також спрацьовує на задачу. Таким чином, під час запуску *Grunt*, отримуємо попередження в режимі реального часу в терміналі під час редагування вихідного коду. На практиці виникне багато невеликих завдань для конкретних цілей, наприклад, створення проекту. Важливою частиною сили *Grunt* є те, що вона приховує від вас багато роботи.

Gulp

Gulp застосовує інший підхід. Замість того, щоб покладатися на конфігурацію та плагіни, ми маємо справу з фактичним кодом. Якщо ви знайомі з *Unix* та те, як працюють конфігураційні файли, вам більш піді йде *Gulp.* В *Gulp* є готові фільтри, потоки, перенаправлення (*pipe*), та інші інструменти для обробки та передачі результатів збірки. Враховуючи, що конфігурація є кодом, ви можете легко її зламати, якщо зіткнетеся з проблемами. Порівняно з *Grunt*, ви маєте чіткіше уявлення про те, що відбувається.

Browserify

Tusk runers - чудові інструменти на високому рівні. Вони дозволяють виконувати операції крос-платформним способом. Проблеми починаються тоді, коли потрібно з'єднати різні активи разом і створити пакети. З цієї причини існують групи постачальників, такі як *Browrify, Brunch* або *Webpack*, і вони працюють на нижчому рівні абстракції. Замість роботи з файлами вони працюють над модулями та активами. Робота з модулями *JavaScript* завжди була чималою проблемою. Сама мова не мала концепцію модулів до *ES2015*. Мова застрягла в 90-х, коли справа стосується середовища браузера. Запропоновано різні рішення, включаючи *AMD. Browserify* - це одне рішення проблеми модульності. Воно дозволяє об'єднувати модулі *CommonJS* разом. Можна підключити його за допомогою *Gulp*, що дасть більше дрібних інструментів перетворення, які дозволяють вийти за рамки основного використання. Наприклад, *Watchify* надає інструмент для перегляду транспільованого коду, що економить час. Екосистема *Browrify* складається з безлічі маленьких модулів. Таким чином, *Browserify* дотримується філософії *Unix.* З *Browserify* легше почати роботу, ніж з *Webpack*, і насправді є хорошою альтернативою.

Webpack

Webpack - це постачальник модулів. *Webpack* може піклуватися про поєднанням різних пакетів разом з виконанням завдань. Однак, лінія між постачальником та виконанням завдань розмилася завдяки розробленим спільнотою плагінам для *Webpack*. Іноді ці плагіни використовуються для виконання завдань, які зазвичай виконуються за межами веб-пакету, наприклад, для очищення каталогу збірки або розгортання збірки. Реагування на зміни в коді та гаряча заміна модулів (*HMR*) допомогли популяризувати веб-пакет і призвели до його використання в інших середовищах, таких як *Ruby on Rails.* Незважаючи на свою назву, *webpack* не обмежується лише інтернетом. Він також може поєднуватися з іншими завданнями

Огляд реалізації підтримки модулів в Webpack

Найменший проект, який ми можемо поєднати з веб-пакетом, працює з файловою системою. Процес поєднання починається з визначення користувачем пакетів. Самі пакети є модулями і можуть вказувати на інші модулі через імпорт. Коли ми зв'язуємо проект за допомогою *webpack,* він обходить всі імпорти, будуючи (*AST*) дерево залежностей, а потім генерує вихід на основі конфігурації. Крім того, можна визначити розділені точки для створення окремих пакетів у межах самого коду проекту. *Webpack* підтримує формати модулів *ES2015, CommonJS* та *AMD*. Механізм завантажувача працює і для *CSS*, з підтримкою *@import* та *url()* через *css-loader*. Ми також можемо знайти плагіни для конкретних завдань, таких як мінімізація, інтернаціоналізація, *HMR* тощо.

2.4 Детальний огляд інструменту для збірки *Webpack*

Процес виконання Webpack. Обробка модулів програми, створення банднів

На рис. 2.5 показано, як webpack збирає частини проекту (код, статичні ресурси) в бандли (webpack bundles) після чого це попадає в браузер користувачеві по *https\wss*

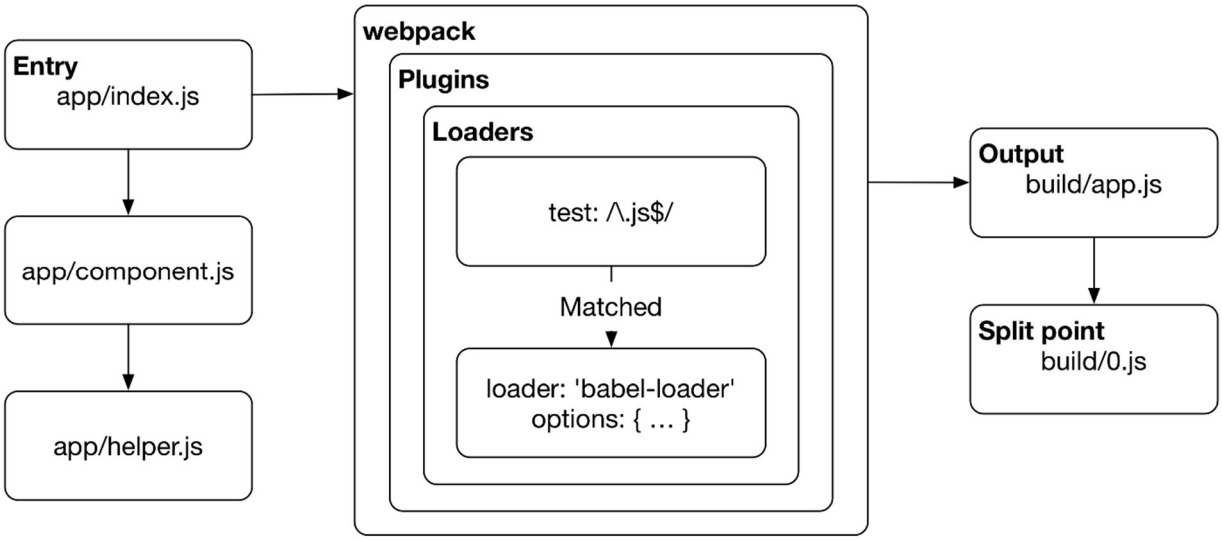


Рис 2.5 Ілюстрація роботи збірки *Webpack*

Webpack починає свою роботу з записів. Часто це модулі *JavaScript*, де *webpack* починає свій процес обходу. У ході цього процесу *webpack* оцінює відповідність вхідних даних по конфігурації завантажувача, які розповідають *webpack*, як трансформувати кожну відповідність.

Процес розширення

Сам запис - це модуль. Коли веб-пакет зустрічається, веб-пакет намагається спів ставити запис із файловою системою, використовуючи конфігурацію дозволу запису. Ми можемо сказати *webpack* виконати пошук на додаток до конкретних каталогів та додаток до *node_modules*. Можна також відрегулювати спосіб збігу *webpack* з розширеннями файлів, і ми можемо визначити конкретні псевдоніми для каталогів. Розділ "Споживчі пакунки" детальніше висвітлює ці ідеї.

Якщо пропуск роздільної здатності не вдався, *webpack* створює помилку виконання. Якщо веб-упаковці вдалося правильно вирішити файл, *webpack* виконує обробку відповідного файлу на основі визначення завантажувача. Кожен завантажувач застосовує певну трансформацію щодо вмісту модуля. Спосіб завантаження цільового файлу може бути налаштований декількома способами, включаючи тип файлу та місцезнаходження у файловій системі. Гнучкість *Webpack* дозволяє навіть застосувати певне перетворення на основі того, де він був імпортований у проект. Такий самий процес роздільної здатності виконується щодо завантажувачів *webpack.*

Webpack дозволяє застосовувати аналогічну логіку при визначенні того, яким завантажувачем він повинен користуватися. З цієї причини навантажувачі мають власну конфігурацію вирішення. Якщо веб-пакет не виконає пошук завантажувача, це призведе до помилки виконання.

Webpack файли

Webpack будує графік залежності. Якщо запис містить залежності, процес буде виконуватися рекурсивно, для кожної залежності до тих пір, поки обхід дерева не завершиться. *Webpack* може виконувати цей процес для будь-яких типів файлів, на відміну від спеціалізованих інструментів, таких як транспілятори *Babel* або *Sass*.

Webpack надає нам контроль над тим, як поводитися з різними активами, з якими він стикається. Наприклад, ми можемо вбудовувати активи до своїх пакетів *JavaScript*, щоб уникнути запитів. *Webpack* також дозволяє використовувати методи, такі як модулі *CSS,* щоб поєднати стилізацію з компонентами, а також уникнути проблем зі стандартним стилем *CSS*. Ця гнучкість - це те, що робить *webpack* настільки цінним.

Хоча *webpack* використовується в основному для згрупування JavaScript, він може працювати з об’єктами, такими як зображення або шрифти, і випускати окремі файли для них. Записи є лише відправною точкою процесу збору. Те, що випускає веб-пакет, повністю залежить від способу його налаштування.

Заміна гарячого модуля (HMR)

Інструменти *LiveReload* або *BrowserSync* автоматично оновлюють сторінку під час внесення змін. Можливість гарячої заміни модулів (*HMR*) робить все на крок далі. У випадку *React,* це дозволяє додатку підтримувати свій стан без примусового оновлення.

HMR також доступний у *Browserify* через *livereactload*, тому це не є ексклюзивною функцією веб-пакету.

Розщеплення коду

Webpack дозволяє розділити код різними способами. Ми навіть можемо динамічно завантажувати код, коли наша програма виконується. Таке пасивне навантаження корисне, особливо для ширших застосувань, оскільки пакети можна завантажувати на льоту за потребою.

Навіть невеликі програми можуть отримати вигоду від розщеплення коду, оскільки це дозволяє користувачам швидше завантажити сторінку, що в свою чергу підвищує продуктивність програми.

Огляд фреймворку (framework) ReactJS

ReactJS є *JavaScript* бібліотекою, яка використовується при розробці веб-додатку для створення інтерактивних елементів на веб-сайтах. Чому *JavaScript* розробники використовують *React*?

ReactJS є *JavaScript* бібліотекою, яка спеціалізується на наданні допомоги розробникам створювати користувацькі інтерфейси, або *UIs*. З точки зору веб-сайтів і веб-додатків, *UIs* є збір компонентів на екрані: меню, пошук барів, кнопки, і все інше, з чим хтось взаємодіє використовуючи веб-сайт або додаток.

Віртуальний document object model (DOM)

Якщо ви не використовуєте *React JS*, ваш веб-сайт використовує *HTML*. Це чудово працює для простих, статичних веб-сайтів, але для динамічних веб-сайтів, які передбачають важку взаємодію з користувачем, це може стати проблемою, оскільки весь *DOM* потребує перезавантаження кожного разу, коли користувач викликає функцію, яка змінює сторінку.

Однак, якщо розробник використовує *JSX* для маніпулювання та оновлення своєї *DOM, React JS* створює *Virtual DOM. Virtual DOM*, як випливає з назви - це копія *DOM* сайту, *React JS* використовує цю копію, щоб побачити, які частини фактичного *DOM* потрібно змінити, коли подія відбувається (наприклад, користувач натискає кнопку).

Скажімо, користувач вводить коментар у форму публікації блогу і натискає кнопку "Відправити". Без використання *React JS* весь *DOM* повинен був би оновитись щоб відобразити цю зміну (використовуючи час та ресурси на обробку, необхідні для цього оновлення). З іншого боку *React* сканує *Virtual DOM* щоб побачити, що змінилося після дії користувача (у цьому випадку додається коментар) і вибірково оновиться лише цей вузол *DOM*.

Таке селективне оновлення вимагає менше обчислювальної потужності та менше часу на завантаження.

ВИСНОВОК ДО 2 РОЗДІЛУ

В даному розділі було оглянуто ключові web технології а саме:

- Протокол *HTTP* на якому зараз будуються більшість web додатків.
- *PHP* серверна мова програмування, яка дозволяє реалізувати серверну частину web додатку.
- Системи змірки та виконанню зарані заготованих завдань (*task runners*)
- Webpack система для виконання завдань і не тільки.

Розглянуто історію розвитку цих технологій на основі чого робився вибір в сторону тої чи іншої технології\ продукту\ пакету

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Визначення вимог і завдань

Необхідно реалізувати функціонал який дозволить вносити:

- Статті
- Наукові видання
- Наукові посібники для співробітників

3.2 Моделювання та аналіз програмного забезпечення

KPI Connect - освітня система для університетів. Вона допомагає в зберіганні, модифікації і отриманні даних для різних функцій. Система забезпечує повний доступ до персональних даних, як для адміністрації, викладачів, так і для студентів. Для зв'язку між студентами і викладачами використовується Телеграма *API. KPI Connect* написаний в *Symfony* 3.4, використовує *Sonata Admin Bundle 3.33* для інтерфейсу *Admin* і *ReactJS* для інших користувачів. База даних - *PostgreSQL.* Докер використовується для розробки.

Список технологій:

- *PHP*
- *Sympfony*
- *Docker*
- *ReactJS*
- *PostgreSQL*

3.3 Інструкція початку роботи

Ця інструкція наддасть вам копію проекту на вашій локальній машині для розробки і тестування. Про те, як розгорнути проект на живій системі, див. В розділі "Розгортання".

Встановлення

1. Клонуйте проект у свій каталог *git clone https://gitlab.com/kpi-connect/kpi-connect.git*
2. Перейдіть до папки проекту *cd kpi-connect*
3. Для встановлення всіх пакетів *composer install*
4. Запустити *npm* і в корені проекту
5. Запустити *gulp dev* в корні проекту (*npm i gulp-cli -g*, якщо у вас його ще немає)

Перший запуск

Mac OS

1. Встановіть *Docker* і *Docker-sync*
2. Перейдіть в директорію *cd dev/docker*
3. Початок синхронізацію докера *docker-sync start*
4. Запуск проекту \ міграції бази даних *docker-compose up*

Linux

1. Встановіть *docker-io і docker-compose*
2. Перейдіть в директорію *cd dev/docker*
3. Запуск проекту \ міграції бази даних *docker-compose up*

Windows 10 *build* 2004 і вище

1. *Потрібно встановити WSL 2 (Windows subsystem for Linux v2)*

   - Відкрити «Додаткові компоненти системи *Windows*»
   - Підключити компонент «Платформа віртуальної машини»
   - Перезавантажити систему
   - Перейти в *Microsoft store* та встановити *Ubuntu*
   - Після встановлення *Ubuntu* з’явиться у вас на машині

2. Тепер достатньо виконати інструкцію, яка описана вище для *Linux*

3. Дуже важливо розуміти що *WSL2* на відміну від *WSL1* має свій *IP* адрес який потрібно використовувати замість *localhost* хоча він також працює але бувають винятки. Подивитись *IP* можна так *$ hostname -I | awk '{print $1}'*

3.4 Структура даних в базі даних

На рис. 3.1 показано схему із бази даних та їх взаємодію

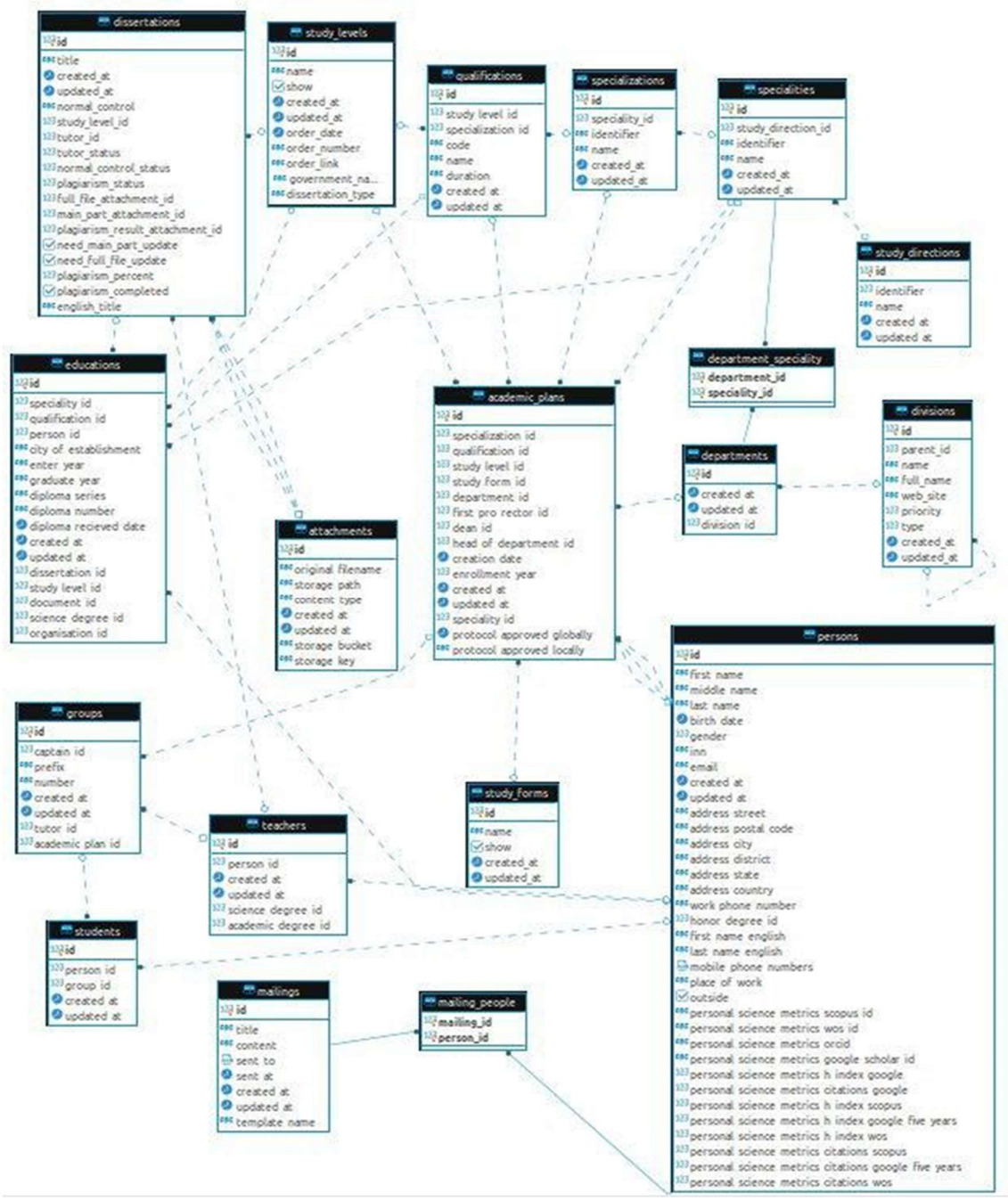


Рис. 3.1 структура бази даних

Наведена вище схема бази даних це фрагмент (тільки та частина, що використовується для розробки) схема раніше розробленого основного сервера *KPI-CONNECT*, ця схема була дещо доповнена для цієї розробки.

Нище наводиться опис що саме містить в таблицях (таблиці зв’язування не описуються):

- *Perons* - містить дані людей (студентів та викладачів);
- *Students* – містить студентів, що пов’язані з деякою персоною та групо.
- *Teachers* – містить викладачів, що пов’язані з деякою персоною та групою як куратор.
- *Devisions* – містить факультети та кафедри.
- *Academic_plans* – містить академічні плани.
- *Groups* – містить групи студентів.
- *Study_forms* – містить форми навчання.
- *Dissertations* – містить поля характеристик випускних робіт студентів.
- *Educations* - містить опис освіти студентів, що пов’язана з дисертацією та студентом.
- *Attachments* – містить документи що відносяться до випускних робіт студентів (повна випускна робота, головна частина (ПЗ) та звіт перевірки).
- *Mailings* – містить повідомлення що будуть відправлятись за розкладом.

ВИСНОВОК ДО 3 РОЗДІЛУ

В цьому розділі було зібрано та систематизовану всю необхідну інформацію для створення веб додатку, створено, встановлено необхідне програмне забезпечення для реалізації додатку на обраній платформі .

РОЗДІЛ 4
ІНСТРУКЦІЯ З ЕКСПЛУАТАЦІЇ

4.1 Вхід до адмін. панелі

Для доступу до адмін. панелі вам необхідно доступ до інтернету та браузер, що дасть вам змогу знайти сайт *http://admin.kpi-connect.test* та пройти процедуру входу в акаунт.

На рис 4.1 продемонстрована сторінка входу в систему

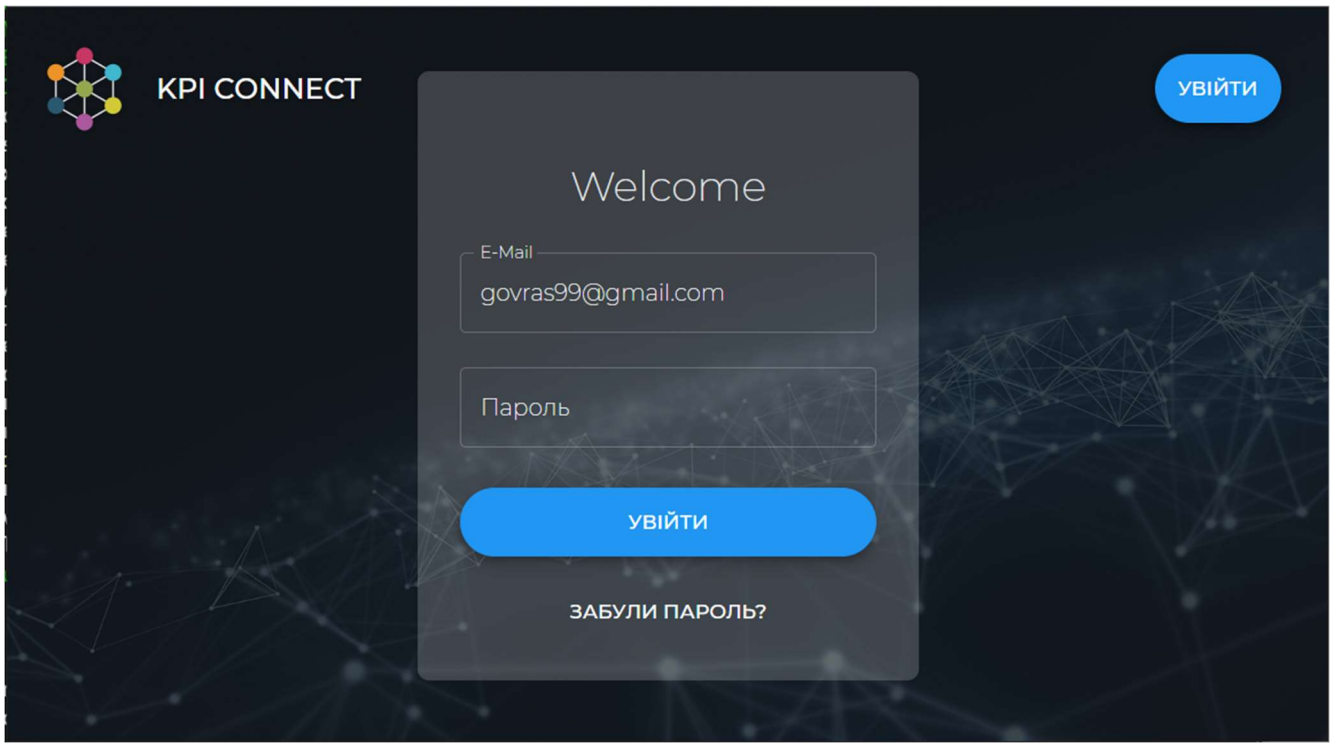


Рис. 4.1 сторінка входу в систему

Для входу необхідно заповнити поля логіна та пароля та натиснути кнопку входу.

4.2 Створення нової персони

Для створення нової персони потрібно увійти в систему з правами адміністратора та виконати наступні кроки:

1. Відкрити вкладку **Кадрові ресурси** приклад на рис. 4.2

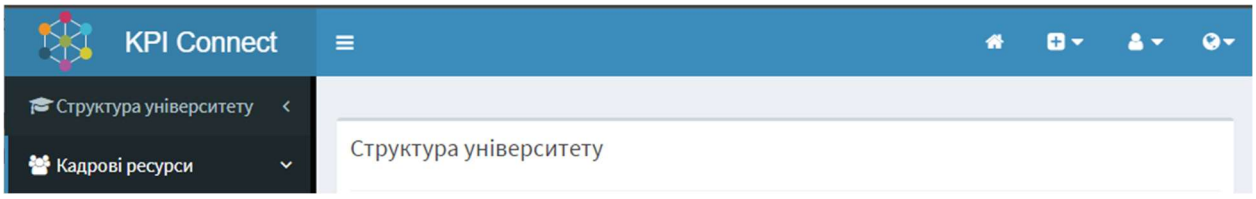


Рис 4.2 Кадрові ресурси

2. В списку обрати пункт **Персона** приклад на рис. 4.3

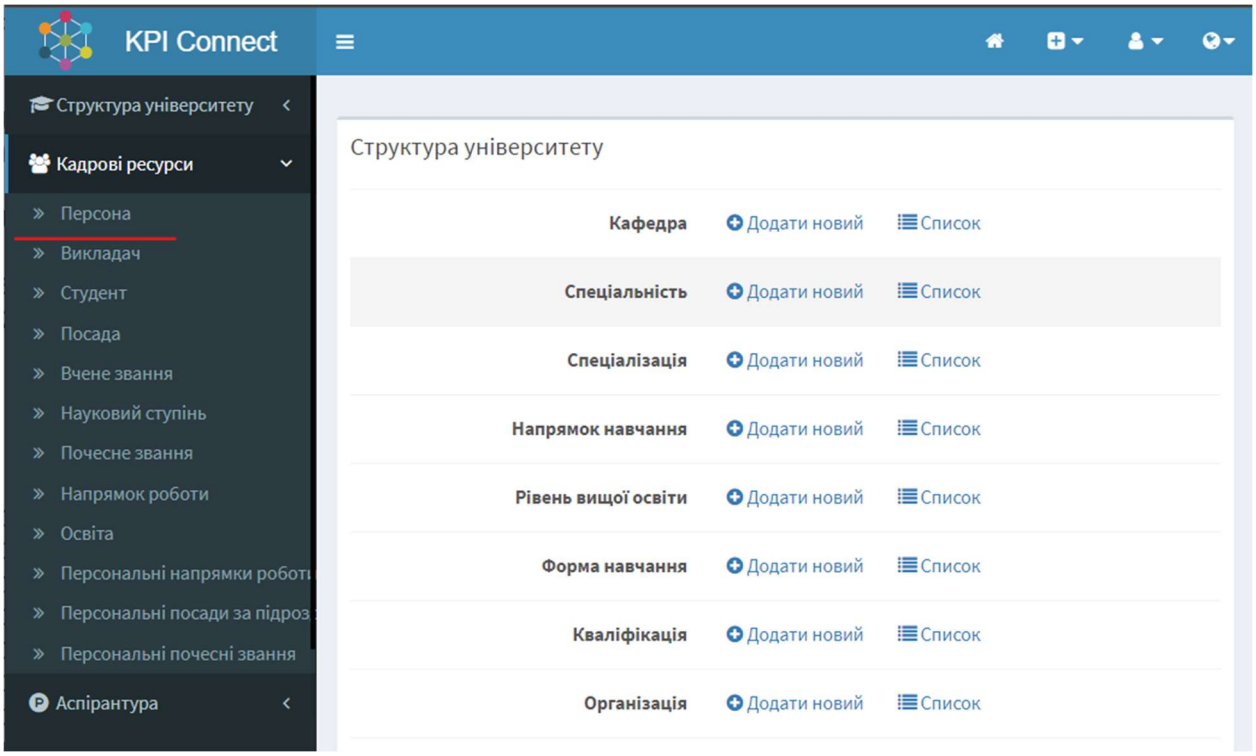


Рис 4.3 Кадрові ресурси

3. Після чого ми зможемо побачити список всіх персон де можна робити пошук та редагувати їх. Приклад показано на рис. 4.4.

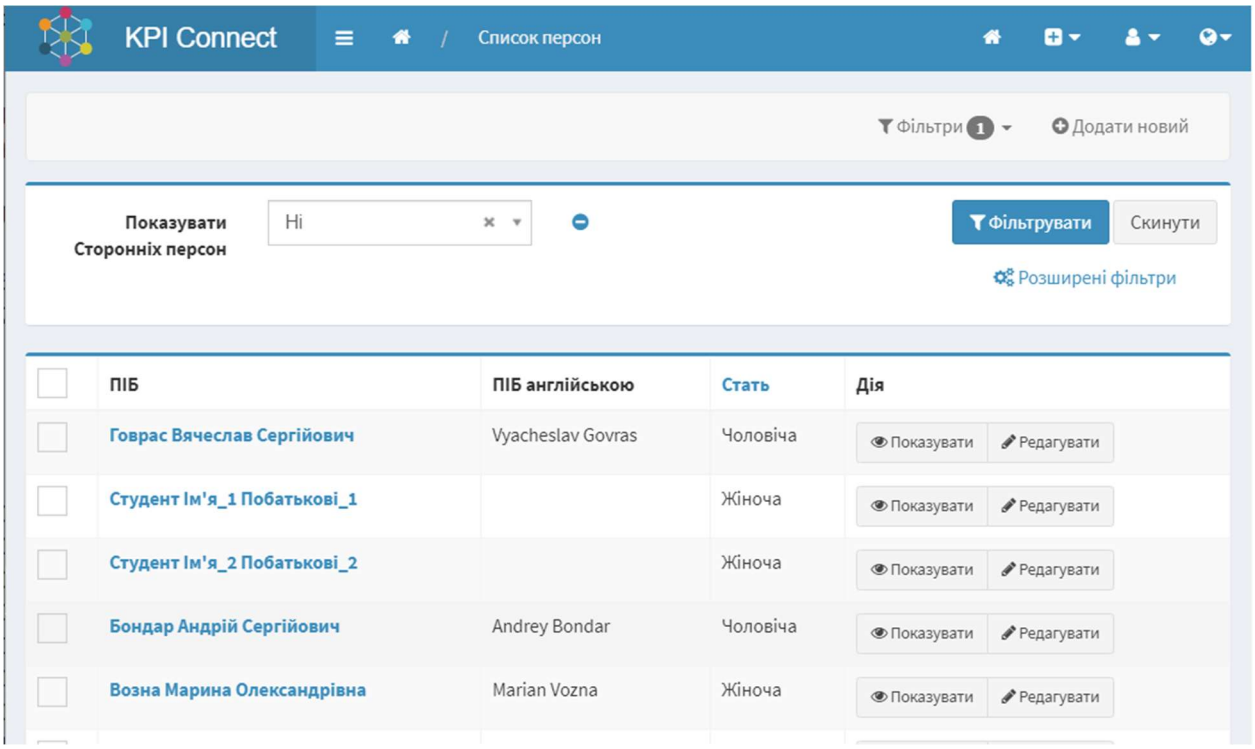


Рис 4.4. Кадрові ресурси

4. Щоб додати нову персону натисніть **Додати новий**. Приклад показано на рис. 4.5.

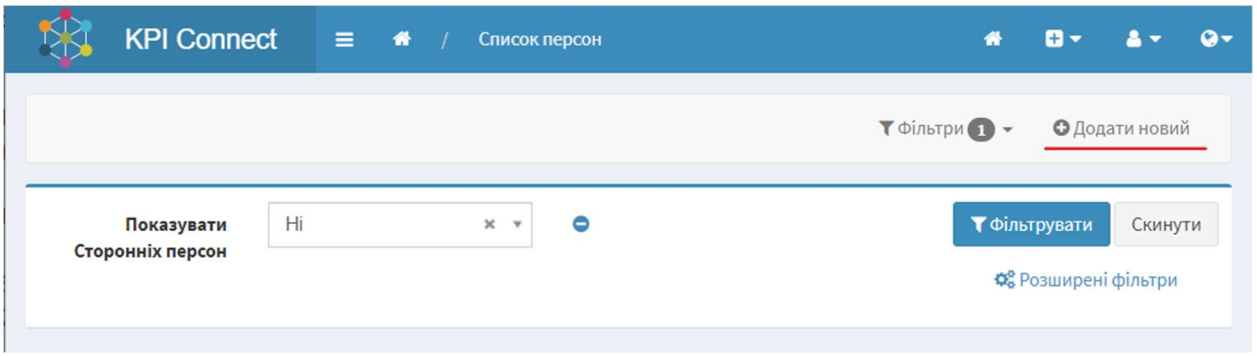


Рис 4.5. Кадрові ресурси

5. Далі ви попадете на форму де потрібно заповнити дані про персону

Дана форма розділена на 4 основні розділи (персональна інформація, контакти, адреса, персональні науково-методичні показники). Приклад можна побачити на рис. 4.6.

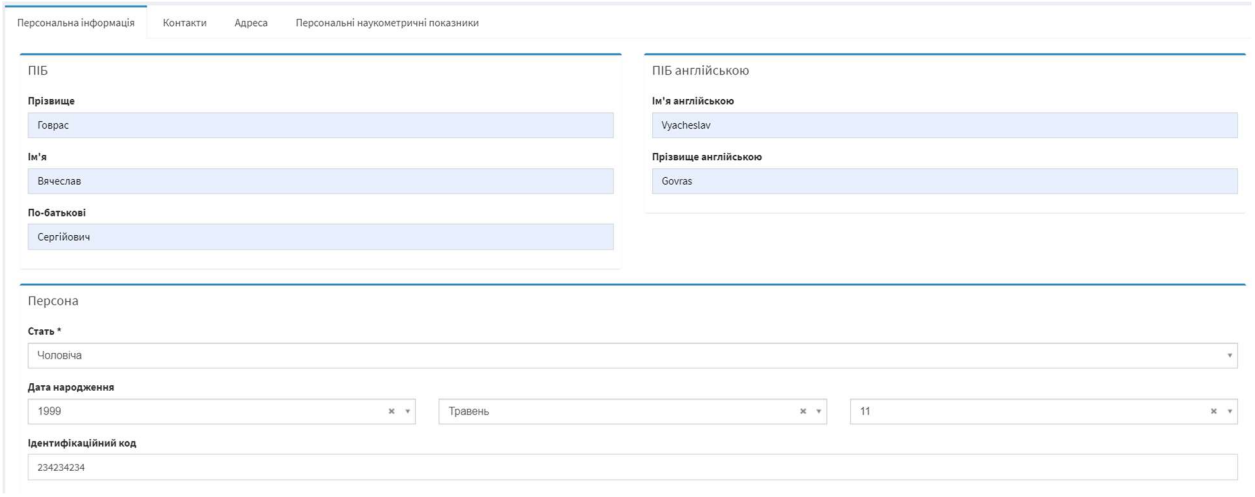


Рис 4.6 Кадрові ресурси

Персональна інформація

Даний таб має 3 форма:

1. ПІБ (Прізвище, ім’я, по-батькові персони так як вказано в документах про дану персону). Приклад продемонстровано на рис. 4.7.

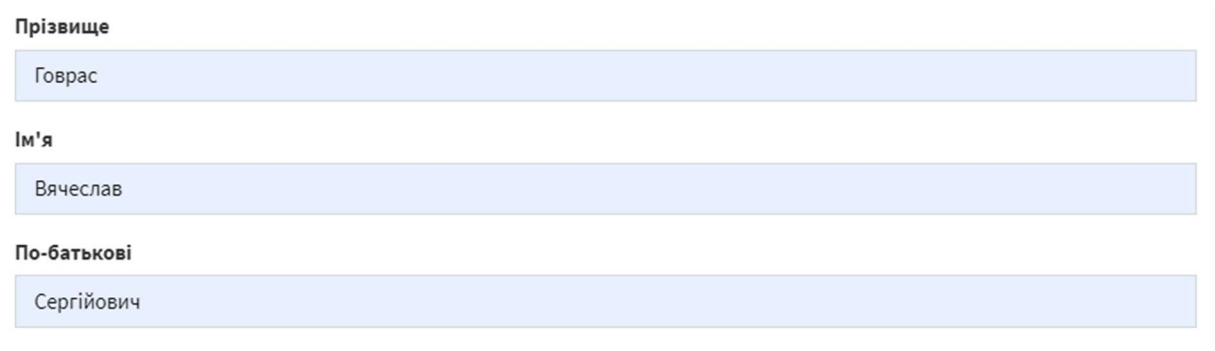


Рис 4.7 Кадрові ресурси

2. ПІБ англійською (Прізвище, ім’я, по-батькові персони на англійській мові). Приклад показано на рис. 4.8.

Рис 4.8 Кадрові ресурси

3. Персона (Стать дата народження та ідентифікаційний код платника податків)

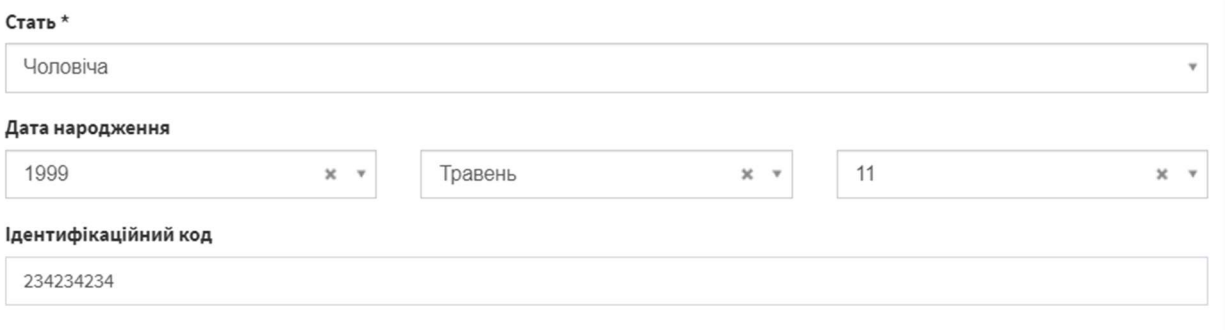


Рис 4.9 Кадрові ресурси

Контакти

Тут потрібно вказати контакту інформацію майбутньої персони, а саме:

- Місце роботи
- Номер мобільного
- Адреса електронної пошти

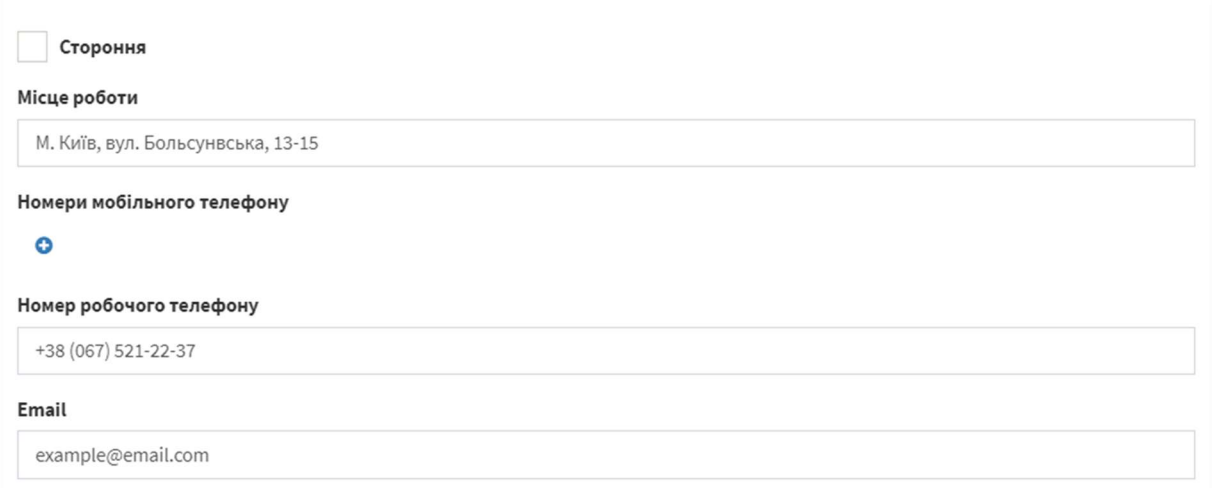


Рис 4.10 Кадрові ресурси

Адреса

В даному табі потрібно вказати місце проживання персони

Рис 4.11 Кадрові ресурси

Персональні науково-методичні показники

Тут перелічені всі можливі показники персони

Рис 4.12 Кадрові ресурси

4.3 Інструкція створення наукового видання

Пройшовши процедуру входу ви можете увійти до адмін. панелі.

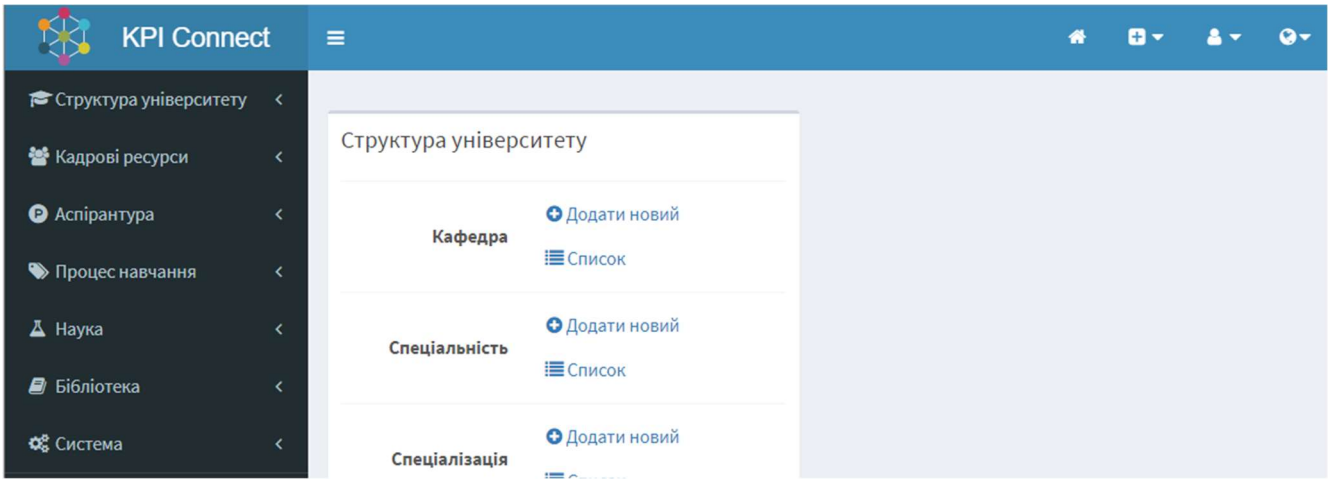


Рис. 4.13 Знімок амін. Панелі.

Для того щоб додати нове наукове видання потрібно:

1. Обрати пункт Наука;
2. В списку, який відкриється обрати Наукове видання ;
3. Натиснути + Додати новий;

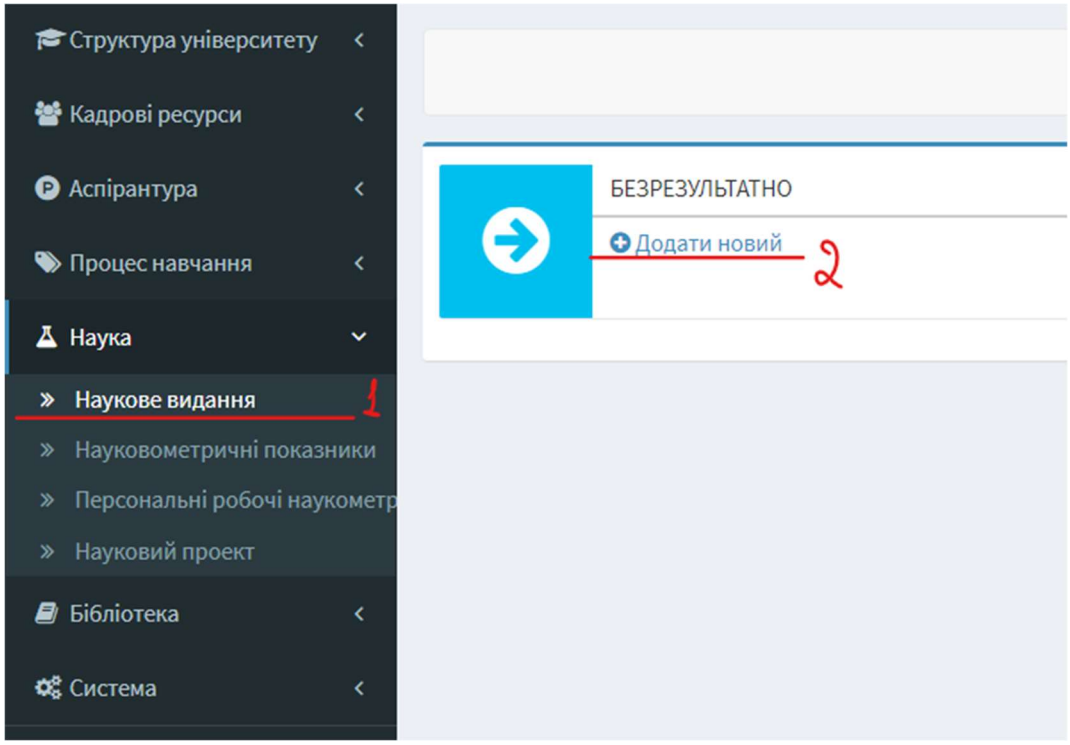


Рис. 4.14 Приклад створення наукової статті.

Після цих дій нам стане доступна форма для створення нового наукового видання. Для його створення потрібно заповнити форму (рис. 4.15)

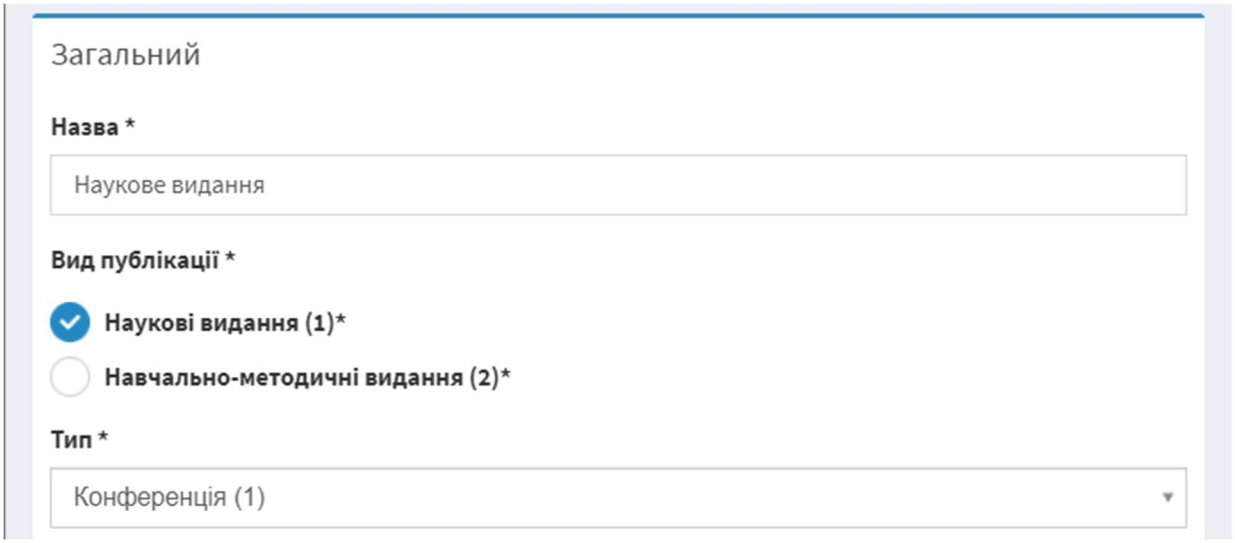


Рис. 4.15 Сторінка налаштувань створення видань.

Назва: ввести назву публікації із поля «Назва роботи».

Вид публікації: Вибрати «Наукове видання».

Тип: (див. в полі «Назва видання, де опубліковано» там зазвичай є слово конференція, якщо немає, то це стаття в журналі):

- Конференція ;
- Стаття;

Мова публікації: див. по назві публікації, якщо англійська, вибираємо англійську.

Місце публікації: (Див. в полі «Назва видання, де опубліковано», зазвичай написано місце проведення конференції, країна видання журналу):

- Країни ЄС (Європа, Америка);
- Інші країни (Совок, Росія, Казахстан, …);
- Українські фахові (зазвичай всі українські видання і конференції);
- Інші (науково-публіцистичні, в цьому списку таких немає);

Науко метричне джерело публікації: В файлі три розділи, спочатку ***Scopus***, вибираємо відповідний пункт списку, потім ***Web of Science***, потім **Інші.**

Закордонна публікація: (Див. вище) Проставити прапорець, якщо вибрали перший або другий пункт в «Місце публікації».

***Impact Factor, SNIP*>=0,4** – не проставляєм, це не часто буває.

Друковане видання: Для конференцій зняти прапорець.

Коментар: Скопіювати авторів, як є в полі АВТОРИ

Анотація: Нічого.

Вихідні дані:

- Назва видання (журналу) або конференції Для конференції: ***Обов'язково!;***
- Місце і дата проведення конференції із поля «Назва видання, де опубліковано», для статті в журналі нічого
- Номер тому, номер журналу, наприклад: ***vol 754****, або* ***Vol 2, No 9 (92)*** *DOI*:, якщо є;
- Посилання на статтю (Якщо немає *DOI*) . Якщо знайдете посилання на статтю, або журнал, або конференцію з цією публікацією, сюди внести посилання.;
- Краще за все кинути назву конференції в Гугл, знайти архів, зміст, якщо стаття не доступна розмістити посилання на зміст. Заодно продивитись де і коли проходила, не у всіх це вказано. Якщо є *DOI*, прямо в Гугл вбивати і по цьому номеру можна знайти статтю і конференцію і всю інформацію про видання сторінки, тощо ;

Сторінки: Наприклад, 50-67.

Видавництво: Виділено голубим в списку, або якщо знайдете, там написано, що уточнити. Для журналів повинно бути обов'язково, якщо не знайдете, то хоча б країну. Для конференцій, необов'язково, тільки якщо вона друкована, це рідко.

Автори: Додаємо всіх авторів по одному. Треба перекладати прізвища українською і вбивати у поле. Там є пошук, відразу випадає, хто є в базі. Кого немає не вносимо. Авторів, кого немає в базі виділити в списку сірим.

Рік видання: вказати рік коли публікацію було видано.

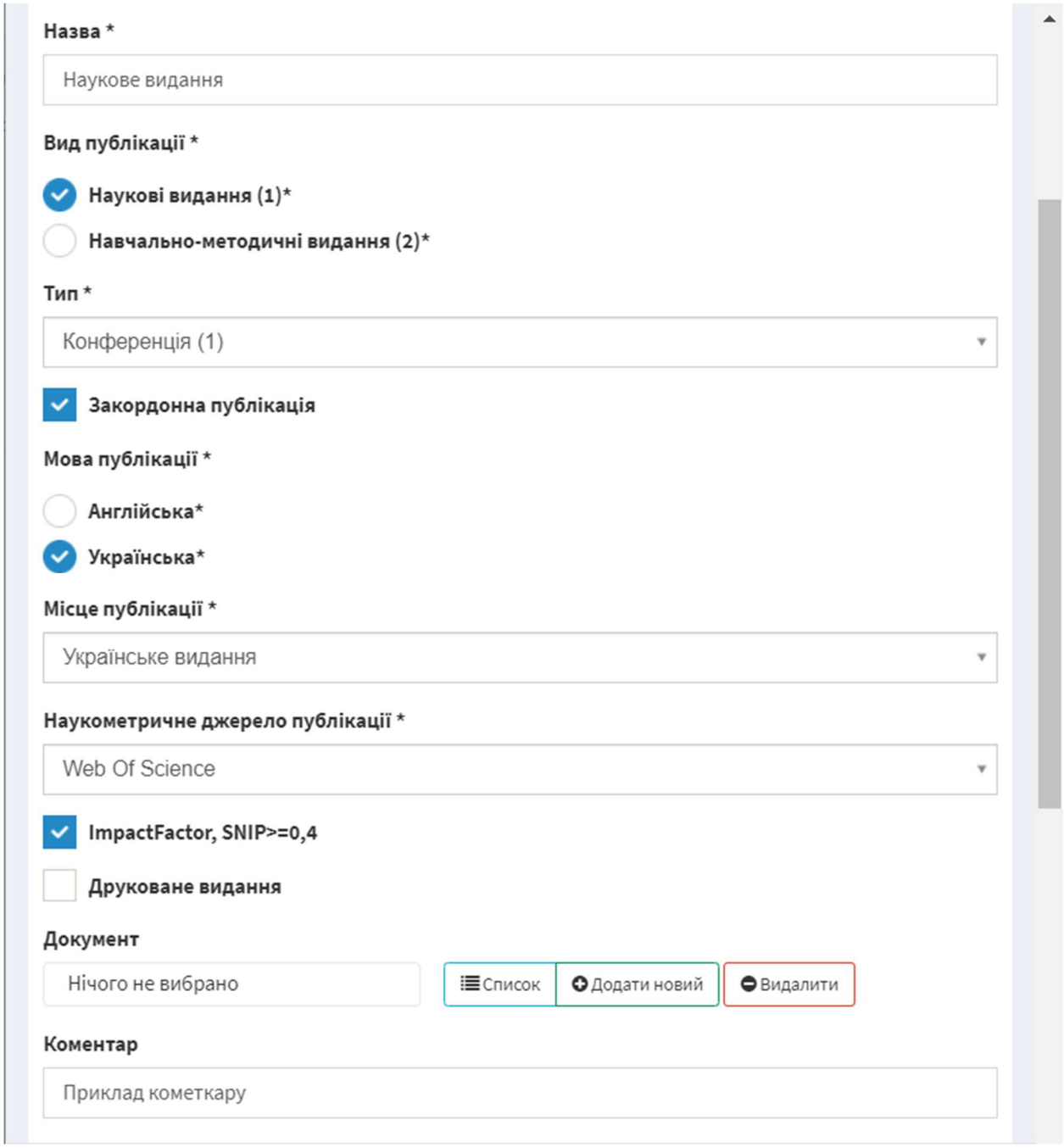


Рисунок 4.16 Приклад заповненої форми

Після створення ми повернимось до списку де можна побачити наше наукове видання, після чого можна виконати з ним якісь дії *(редагувати / переглянути)*.

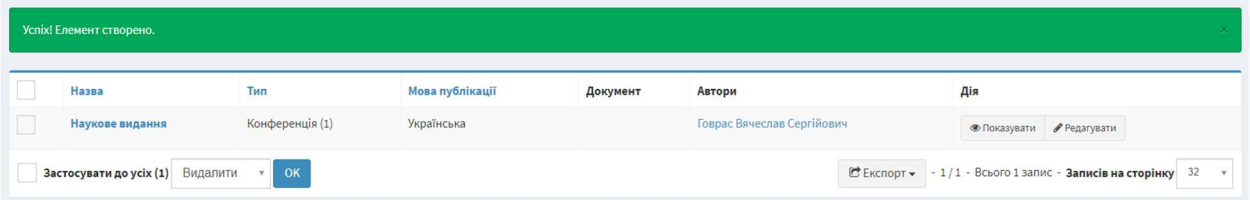


Рисунок 4.17 Приклад результату успішного створення видання.

ВИСНОВОК ДО 4 РОЗДІЛУ

В даному розділі було розроблено документацію для розробленої системи електронного забезпечення науково-методичної діяльності вищого навчального закладу. Описано всі технічні нюанси. Створено детальну інструкцію користувача в якій пописано функціональні можливості та способи взаємодії з системою. Продемонстровано виконання основних системних функцій, а саме:

- Створення нової персони
- Заповнення інформації, яка з нею пов’язана
- Створення нового наукового вивчення
- Створення Навчально-методичного видання

ВИСНОВКИ

Темою даної роботи було створення “Системи електронного забезпечення науково-методичної діяльності вищого навчального закладу” яка дозволить здійснювати зберігання та обробку науково-методичної та навчально-методичної документації, яка буде закріплення за конкретною персоною для подальшого пошуку та її обробки.

В даній роботі було поставлено розробити систему, описано основні вимоги та функціональні можливості, розглянуто технології які присутні на ринку продемонстровано їх переваги та недоліки на основі, чого обрано платформу та інструменти для подальшої розробки.

В роботі описано всі технологічні аспекти даної системи, та способи її підтримки. розглянуто створенні функціональні можливості та задокументовано в детальній інструкції користувача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Програмування. PHP: створення динамічних веб-сторінок / Peter MacIntyre, Kevin Tatroe. O'Reilly Media; 3-тє видання 2013р
2. Створення одно сторінкових веб-додатків: JavaScript від початку до кінця. = Single Page Web Applications: JavaScript end-to-end 1st Edition / [Josh Powell, Michael Mikowski.] Manning Publications; 2013
3. MongoDB vs. MySQL [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.mongodb.com/compare/mongodb-mysql
4. About Docker IBM [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/docker
5. GitHub code source [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://github.com/docker
6. Docker open source project [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/containerization
7. Специфікація протокол HTTP [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc2616
8. Документація по Node.JS [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://nodejs.org/en/abou

ДОДАТОК А. КОД ПРОГРАМИ.

Приклад розробленого ендпоінту getAction

```
<?php

namespace KPIConnectBundle\Controller\Api\Teacher\Diploma;

use Doctrine\Common\Collections\ArrayCollection;
use FOS\RestBundle\Controller\Annotations as Rest;
use FOS\RestBundle\Controller\FOSRestController;
use KPIConnectBundle\Model\Domain\Dissertation\DissertationTransformer;
use KPIConnectBundle\Services\Diploma\TutorDiplomaService;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Security\Core\Security;
use KPIConnectBundle\Annotation\ApplyMiddleware;
use KPIConnectBundle\Middleware\TeacherMiddleware;

/**
 * Class GetAction
 **/
class GetAction extends FOSRestController
{
    const STUDY_LEVEL_ID_PARAMETER_NAME = 'studyLevel';
    /**
     * @var DissertationTransformer
     */
```

```
    private $dissertationTransformer;
    /**
     * @var TutorDiplomaService
     */
    private $tutorDiplomaService;

    /**
     * GetAction constructor.
     *
     * @param DissertationTransformer $dissertationTransformer
     * @param TutorDiplomaService $tutorDiplomaService
     */
    public function __construct(DissertationTransformer $dissertationTransformer,
TutorDiplomaService $tutorDiplomaService)
    {
        $this->dissertationTransformer = $dissertationTransformer;
        $this->tutorDiplomaService = $tutorDiplomaService;
    }

    /**
     * @inheritDoc
     *
     * @ApplyMiddleware(TeacherMiddleware::class)
     * @Rest\Get(path="/teacher/diplomas", name="api_teacher_diplomas")
     */
    public function tutorDiplomasAction(Request $request, Security $security)
    {
        $studyLevel = $request->query-
>get(self::STUDY_LEVEL_ID_PARAMETER_NAME);
```

```
        $teacher = $this->getUser()->getPerson()->getTeacher();
        $diplomas = $this->tutorDiplomaService->getStudentsDiplomas($teacher,
$studyLevel);

        if (!$diplomas) {
            return JsonResponse::create([
                'error' => 'diplomas_not_found',
                'error_description' => 'Diplomas is not found'
            ], 404);
        }

        return $diplomas->map(function($diploma) {
            return $this->dissertationTransformer->transform($diploma, [
                'addGroup' => true
            ]);
        });
    }

    /**
     * @inheritDoc
     *
     * @ApplyMiddleware(TeacherMiddleware::class)
     * @Rest\Get(path="/teacher/diplomas/{diplomaId}",
name="api_teacher_diploma", requirements={"diplomaId"="\d+"})
     */
    public function tutorDiplomaAction(int $diplomaId)
    {
```

```
        $teacher = $this->getUser()->getPerson()->getTeacher();
        $diploma = $this->tutorDiplomaService->getStudentsDiploma($teacher,
$diplomaId);


        return $this->dissertationTransformer->transform($diploma, [
            'addGroup' => true,
            'addStudyLevel' => true
        ]);
    }
}
```

Приклад розробленого ендпоінту uploadAction

```
<?php

namespace KPIConnectBundle\Controller\Api\Student\Diploma;

use FOS\RestBundle\Controller\Annotations as Rest;
use FOS\RestBundle\Controller\FOSRestController;
use KPIConnectBundle\Annotation\ApplyMiddleware;
use KPIConnectBundle\Event\DissertationEvent;
use KPIConnectBundle\Model\Domain\Dissertation\Dissertation;
use KPIConnectBundle\Model\Domain\Dissertation\DissertationTransformer;
use KPIConnectBundle\Services\Domain\StudentService;
use KPIConnectBundle\Services\Diploma\TutorDiplomaService;
use KPIConnectBundle\Services\Diploma\Util\DiplomaUploader;
use Symfony\Component\EventDispatcher\EventDispatcherInterface;
use Symfony\Component\HttpFoundation\File\UploadedFile;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use KPIConnectBundle\Middleware\StudentMiddleware;
use KPIConnectBundle\Middleware\StudentGraduateMiddleware;

/**
 * Class UploadAction
 *
 * @package KPIConnectBundle\Controller\Api\Student\Diploma
*/
```

```
class UploadAction extends FOSRestController
{    const FULL_FILE_NAME = 'full';
    const MAIN_PART_FILE_NAME = 'main_part';

    const FILE_SIZE_LIMIT = 1024 * 1024 * 150;
    const FILE_ALLOWED_MIME_TYPES = [
        'application/msword',
        'application/vnd.openxmlformats-
officedocument.wordprocessingml.document',
    ];

    const PDF_MIME_TYPES = [
        'application/pdf'
    ];

    /**
     * @var DissertationTransformer
     */
    private $dissertationTransformer;

    /**
     * @var TutorDiplomaService
     */
    private $tutorDiplomaService;

    /**
     * @var StudentService
     */
    private $studentService;
```

```
/**
 * @var DiplomaUploader
 */
private $diplomaUploader;

/**
 * @var EventDispatcherInterface
 */
private $dispatcher;

/**
 * UploadAction constructor.
 * @param DissertationTransformer $dissertationTransformer
 * @param TutorDiplomaService $tutorDiplomaService
 * @param StudentService $studentService
 * @param DiplomaUploader $diplomaUploader
 * @param EventDispatcherInterface $dispatcher
 */
public function __construct(
    DissertationTransformer $dissertationTransformer,
    TutorDiplomaService $tutorDiplomaService,
    StudentService $studentService,
    DiplomaUploader $diplomaUploader,
    EventDispatcherInterface $dispatcher
)
```

```
{
        $this->dissertationTransformer = $dissertationTransformer;
        $this->tutorDiplomaService = $tutorDiplomaService;
        $this->studentService = $studentService;
        $this->diplomaUploader = $diplomaUploader;
        $this->dispatcher = $dispatcher;
    }



    /**
     * @inheritDoc
     *
     * @ApplyMiddleware({StudentMiddleware::class,
StudentGraduateMiddleware::class})
     * @Rest\Post(path="/student/diploma/upload", name="api_student_upload")
     */
    public function diplomaUploadAction(Request $request)
    {
        $person = $this->getUser()->getPerson();
        $dissertation = $this->studentService->getDiploma($person);


        if (!$dissertation) {
            return JsonResponse::create([
                'error' => 'diploma_not_found',
                'error_description' => 'Diploma not found'
            ], 404);
        }
```

```
        if (!$request->files->has(self::FULL_FILE_NAME) && !$request->files->has(self::MAIN_PART_FILE_NAME)) {
            return JsonResponse::create([
                'error' => 'bad_request',
                'error_description' => 'Full or main part diploma should be sent'
            ], 400);
        }

        /** @var UploadedFile $full */
        if ($full = $request->files->get(self::FULL_FILE_NAME)) {
            if ($full->getSize() > self::FILE_SIZE_LIMIT) {
                return JsonResponse::create([
                    'error' => 'bad_request',
                    'error_description' => 'Full file has more than 150M'
                ], 400);
            } elseif (!in_array($type = $full->getClientMimeType(), array_merge(self::FILE_ALLOWED_MIME_TYPES, self::PDF_MIME_TYPES))) {
                return JsonResponse::create([
                    'error' => 'unsupported_media_type',
                    'error_description' => "Full file has not allowed MIME $type"
                ], 415);
            }

            $this->diplomaUploader->uploadFullDiploma($dissertation, $full);
        }
```

```
        /** @var UploadedFile $mainPart */
        if ($mainPart = $request->files->get(self::MAIN_PART_FILE_NAME)) {
            if ($mainPart->getSize() > self::FILE_SIZE_LIMIT) {
                return JsonResponse::create([
                    'error' => 'bad_request',
                    'error_description' => 'Main part file has more than 150M'
                ], 400);
            } elseif (!in_array($type = $mainPart->getClientMimeType(),
self::FILE_ALLOWED_MIME_TYPES)) {
                return JsonResponse::create([
                    'error' => 'unsupported_media_type',
                    'error_description' => "Main part has not allowed MIME $type"
                ], 415);
            }

            $this->diplomaUploader->uploadMainPartDiploma($dissertation,
$mainPart);
        }

        $this->studentService->afterStudentDiplomaUpload($dissertation);
        if ($dissertation->isUploadComplete() && $dissertation->getTutorStatus()
=== Dissertation::TUTOR_STATUS_PENDING) {
            $this->dispatcher-
>dispatch(DissertationEvent::STUDENT_UPLOAD_COMPLETE, new
DissertationEvent($dissertation));
        }

        return JsonResponse::create([], 202);
    }
```

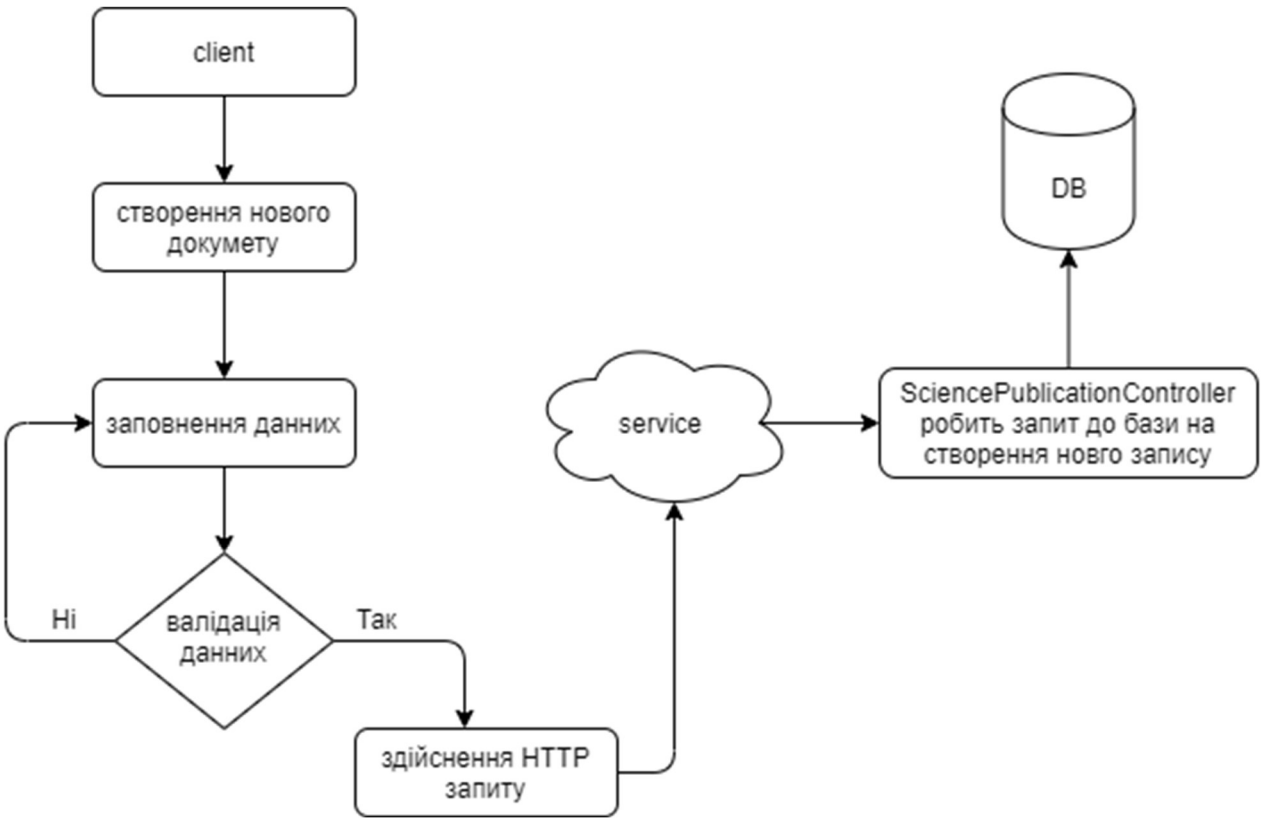
ДОДАТОК 1

«Система електронного забезпечення науково-методичної діяльності вищого навчального закладу»

СХЕМА ВЗАЄМОДІЇ КЛІЄНТА ТА СЕРВЕРА

АРКУШІВ 1

Київ – 2020



					ІАЛЦ.467800.003 Д1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Говрас В.С.			Система електронного забезпечення науково-методичної діяльності вищого навчального закладу	Літ.	Арк. 1 / Акрушів 1
Перевір.		Клименко І.А.					
Н. Контр.		Симоненко				НТУУ «КПІ» ФІОТ ІП-64	
Затверд.		Стіренко С.Г.					

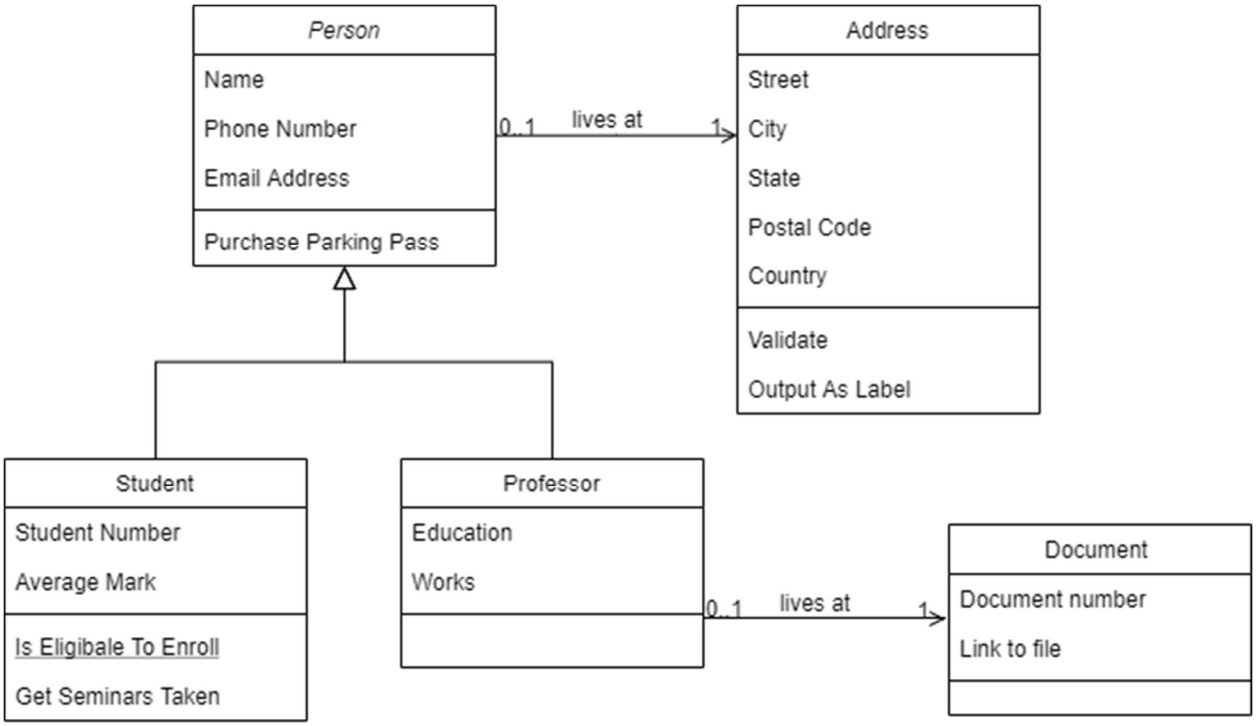
ДОДАТОК 2

«Система електронного забезпечення науково-методичної діяльності вищого навчального закладу»

ДІАГРАМА КЛАСІВ КОРИСТУВАЧА

АРКУШІВ 1

Київ – 2020



					ІАЛЦ.467800.004 Д2	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Говрас В.С.			Система електронного забезпечення науково-методичної діяльності вищого навчального закладу	Літ. / Арк. 1 / Акрушів 1
Перевір.		Клименко І.А.				
Н. Контр.		Симоненко				НТУУ «КПІ» ФІОТ ІП-64
Затверд.		Стіренко С.Г.				

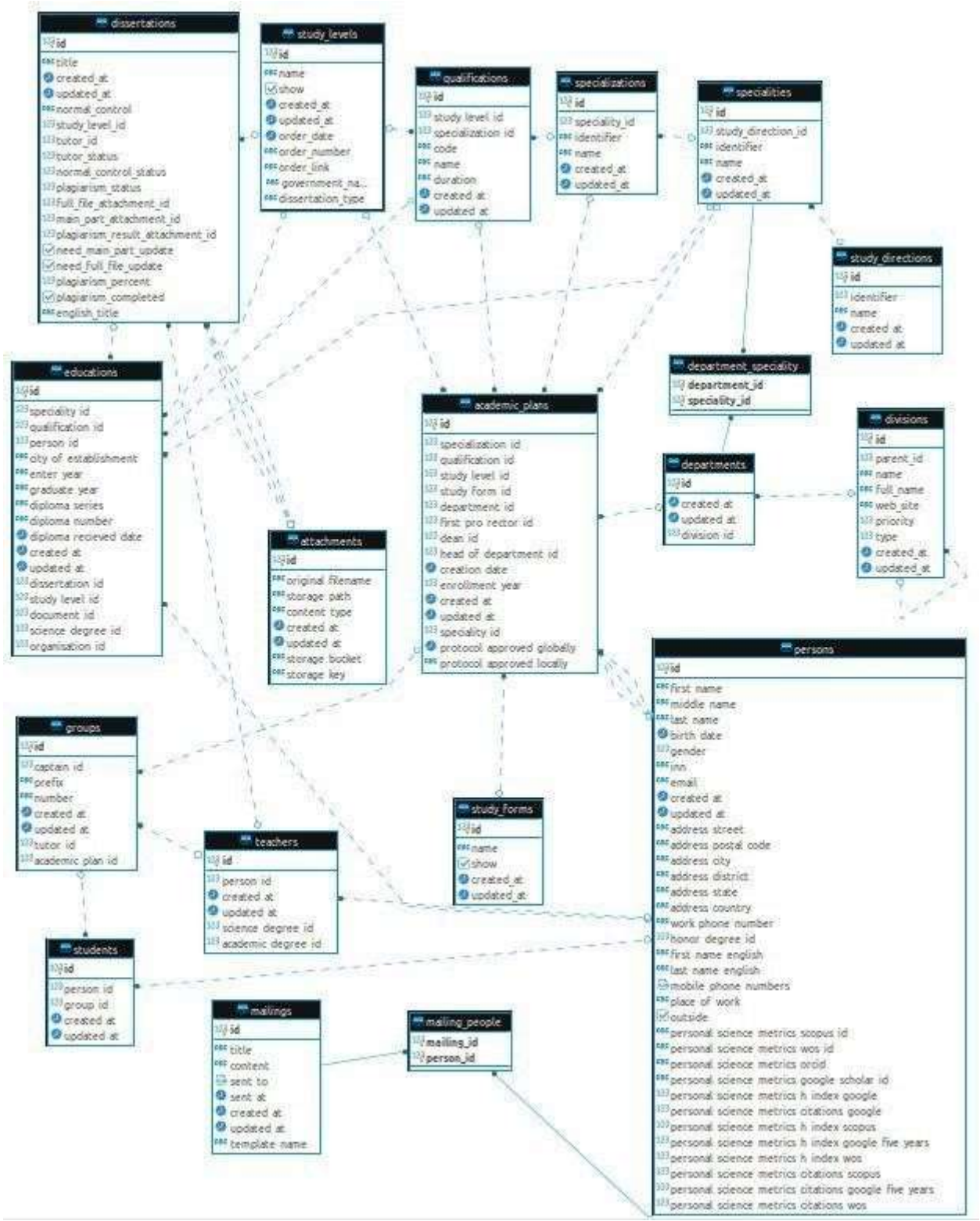
ДОДАТОК 3

«Система електронного забезпечення науково-методичної діяльності вищого навчального закладу»

СХЕМА БАЗИ ДАННИХ

АРКУШІВ 1

Київ – 2020



					ІАЛЦ.467800.005 Д3	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Говрас В.С.			Система електронного забезпечення науково-методичної діяльності вищого навчального закладу	Літ. / Арк. 1 / Акрушів 1
Перевір.		Клименко І.А.				
Н. Контр.		Симоненко				НТУУ «КПІ» ФІОТ ІП-64
Затверд.		Стіренко С.Г.				